AFRL-RI-RS-TR-2008-113
Final Technical Report
April 2008

STATE SPACES FOR CONTINUOUS, SPATIAL, AND ADVERSARIAL BATTLE MANAGEMENT

University of Southern California

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2008-113 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/	/s/
KYLE HOLBRITTER Work Unit Manager	JAMES W. CUSACK, Chief Information Systems Division Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
APR 08	Final	Jun 07 – Sep 07

4. TITLE AND SUBTITLE

STATE SPACES FOR CONTINUOUS, SPATIAL, AND ADVERSARIAL BATTLE MANAGEMENT

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA8750-07-2-0179

5c. PROGRAM ELEMENT NUMBER
N/A

6. AUTHOR(S)

Paul Cohen

5d. PROJECT NUMBER
DEEP

5e. TASK NUMBER
00

5f. WORK UNIT NUMBER
02

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Southern California
University Gardens, Ste 203
Los Angeles CA 90089-0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RISF
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2008-113

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 08-2174

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The Adversarial Continuous Time and Space Search (ACTSS) system helps a commander make better battle decisions by providing detailed, probabilistic models of future battle trajectories given possible courses of action. ACTSS models a noisy world with fluents, thereby allowing us to efficiently capture both the continuous and stochastic nature of real-world operations. Rather than relying on copious amounts of sampling to estimate future outcomes, fluents take advantage of process models that can either be solved in closed-form or can be efficiently updated recursively. We implemented a prediction system using this fluent-based representation, called ACTSS. To test ACTSS, we developed an abstract battlefield simulator, called Arena War. Arena War is a free-for-all melee, where the objective is simply to be the last one standing. The ACTSS system helps the commander to plan operations in Arena War, generating, evaluating, and monitoring possible futures. It identifies potential critical points in these futures, and it heuristically ranks the options for possible next actions.

15. SUBJECT TERMS
Modeling & Simulation, Adversarial Modeling, Simulation Technology
Reinforcement Technology

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Kyle Holbritter
U	U	U	UU	24	19b. TELEPHONE NUMBER (*Include area code*) N/A

Table of Contents

1. Executive Summary **1**

2. The Arena War battle simulator **2**

2.1. Game Model **2**

3. The ACTSS System **3**

3.1. THE FLUENTS……..... **3**

3.1.1. THE TECHNICAL DETAILS..…….…….**4**

3.2. FUTURE GRAPHS…….**5**

3.2.1. Branching in a Futures Graph **6**

3.2.2. Independent vs. Joint Fluents **7**

3.2.3. Updating the Futures Graph **7**

3.2.4. Detailed Example **7**

3.3 CRITICAL POINTS…….....**9**

3.4. HEURISTIC EVALUATION OF GAME STATE……**10**

3.5. INTEGRATION WITH ARENA WAR……..**11**

3.6 DATA AND EXPERIMENTS……...**13**

3.6.1. Commander Success with ACTSS **13**

3.6.2. Efficiency Gains **14**

4. Accomplishments and Lessons Learned **15**

4.1 OPEN QUESTIONS……...**16**

4.2 RELATED WORK……**18**

List of Figures

FIGURE 1. SCREENSHOT OF THE ARENA WAR GAME…….2

FIGURE 2. JOINT DISTRIBUTION OF THE TIME AND THE VICTORIOUS UNIT'S MASS AT THE MOMENT OF VICTORY ……..……...…….6

FIGURE 3. UNIT A AND B ON A BATTLEFIELD, WITH LOCATIONS X, Y, AND Z MARKED ON THE FIELD……..8

FIGURE 4. SEARCH TREE……...8

FIGURE 5. THE THREE TYPES OF CRITICAL POINTS MONITORED..... 10

FIGURE 6. SCREENSHOT OF A FUTURES GRAPH PRESENTED TO THE COMMANDER. DIAMONDS REPRESENT FLUENTS, AND OVALS ARE STATE SNAPSHOTS…….12

FIGURE 7. CRITICAL POINTS ARE DISPLAYED TO THE COMMANDER BY VISUALLY DISPLAYING A SAMPLE FROM THE FUTURE TRAJECTORIES THAT RESULT IN THAT TYPE OF CRITICAL POINT……..12

1. Executive Summary

The ACTSS (Adversarial Continuous Time and Space Search) system helps a commander make better battle decisions by providing detailed, probabilistic models of future battle trajectories given possible courses of action. ACTSS models a noisy world with *fluents*, thereby allowing us to efficiently capture both the continuous and stochastic nature of real-world operations. Rather than relying on copious amounts of sampling to estimate future outcomes, fluents take advantage of process models that can either be solved in closed-form or can be efficiently updated recursively. We implemented a prediction system using this fluent-based representation, called ACTSS. To test ACTSS, we developed an abstract battlefield simulator, called Arena War. Arena War is a free-for-all melee, where the objective is simply to be the last one standing. The ACTSS system helps the commander to plan operations in Arena War, generating, evaluating, and monitoring possible futures. It identifies potential critical points in these futures, and it heuristically ranks the options for possible next actions. Armed with this tool, the commander is able to make better battle decisions.

2. The Arena War battle simulator

Arena War is a simulation of a simplified battlefield occurring in 2D. The game draws its inspiration from the Roman Coliseum and the many battles that were fought within its walls. The game is a free-for-all melee, where the objective is to be the last one standing, either by defeating your opponents, or allowing your opponents to defeat each other. Participants in the battle, which we'll typically refer to as *units*, are represented as circles and each unit has a dynamic mass and area that are directly correlated. Battles occur when two unit's areas intersect and continue until one of the units breaks the encounter or dies.

Figure 1 shows a screenshot of the game. The commander's unit is colored red, and the enemy units are either gray or blue. The commander controls his unit either by pre-loading a plan of attack, or by clicking on a location or unit on the screen to issue a new command corresponding to a movement or attack, respectively.

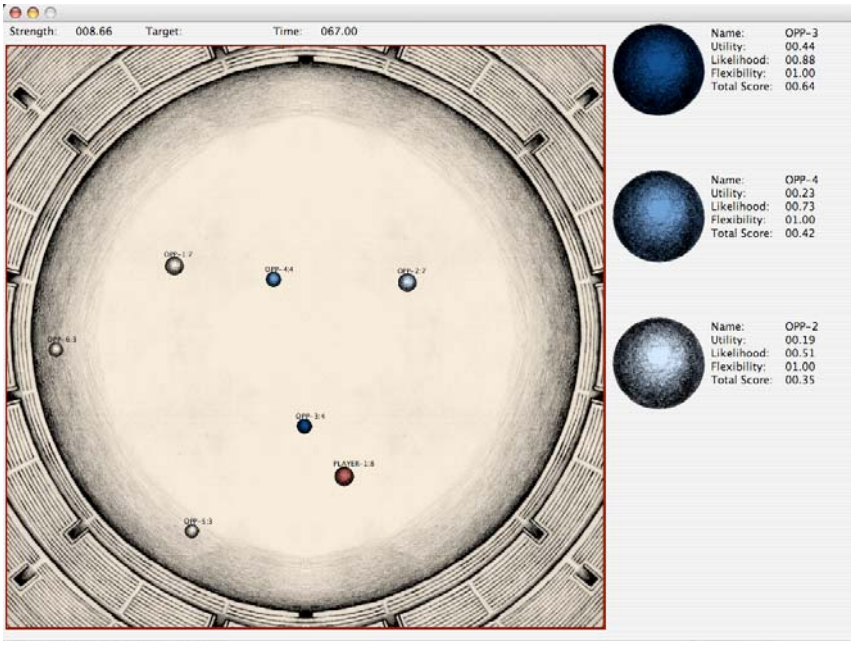


Figure 1. Screenshot of the Arena War game.

2.1. Game Model

Units experience damage from two very different mechanisms. The first and most trivial is damage accrued over time. By analogy, this damage is similar to using fuel with a constantly running engine. As time progresses the amount of mass (fuel) decreases. The second form of damage occurs when two units are engaged in combat. Let m_i and m_j

represent the mass of the *i*th and *j*th unit respectively. While units *i* and *j* are engaged, the probability of actually doing damage to the other unit during the update cycle is determined with a coin flip. The randomness provides a non deterministic element to the outcome of the individual battle and provides an opportunity for a unit with less mass to overcome a larger unit. If unit *i* successfully damages unit *j*, the change in mass is

$$\Delta m_j = m_i * k_i ,$$

based on Lanchester's ``square law'', where *k* is a specified constant. Once a unit falls below the minimum threshold of mass required for continued participation, the unit's mass is distributed equally among the remaining parties that are actively attacking that unit. This introduces new mass into the system and rejuvenates units after battle.

The simulator's update cycle is carried out in several phases. During the first phase each of the units determines the next series of plans they would like to execute (who they would like to attack). Plans are decided for each participant and are ultimately grounded in an actual location. The second phase moves objects that are currently not involved in any engagements. These units could be on the fringe of the battle, or the battle my have just begun, but either way they have yet to clash with another unit. Once all of those units are moved, the simulator calculates all new engagements and freezes the involved unit's velocities. Once we have a list of all actual engagements, the damage round is calculated and areas are updated. After the damage round those objects that haven't moved yet are given the opportunity to move. It is during this phase that engaged units may make an attempt to break engagement.

3. The ACTSS System

Battle operations place great importance on spatial and temporal reasoning. Both time and space are inherently continuous, and our ACTSS system deals with continuous variables using *fluents*. Our main task is to predict the range of probable futures, given a set of plans specified by the commander that describe the actions of our unit and the likely actions of the opponent units. The key part of this task is to identify critical points – situations where the commander lacks good options or alternative courses of action. For example, a situation where our unit becomes surrounded by enemy troops would be a critical point. Our units lacks options; it can only stand and fight. ACTSS generates the range of possible futures given the specified plans, evaluates the futures to determine their desirability, and monitors the futures to identify and alert the commander to critical points.

3.1. Fluents

A fluent models a process of continuous change. Unlike traditional state-based representations, where actions and transitions represent instantaneous changes from one state to the next, a fluent-based representation allows us to capture the continuous change of variables over time. This makes fluents ideal for modeling systems that have a substantial spatial and temporal component, such as battle operations.

Powerful representation is useless unless there is an efficient mechanism for deriving inferences or predictions from the representation. Here we are primarily concerned with having an efficient way of projecting our current state into the future given an action plan. This is non-trivial because of the stochastic nature of the environment and its resultant effect on the interactions of the various units. Small variations in the variables can lead to large eventual effects. Put another way, timing matters --- arriving at a destination just a little late could cost a unit the battle by giving the enemy unit that little bit of extra time to prepare.

Our representation achieves power and efficiency by relying on analytical models of dynamic change, focusing on models that enable us to efficiently propagate estimations of uncertainty into our projected futures. The basic model is a linear system with Gaussian noise. The next section provides an overview of this type of model. With this model, uncertainty about the futures values of state variables can be calculated analytically in polynomial time, rather than requiring large amounts of sampling.

Fluents provide several benefits. First, each fluent encapsulates an episode of interaction between two or more units. This simplifies the search space since the evolution of the state space can be modeled analytically within each fluent. Second, the fluent captures the possible outcomes of the interaction, and can make predictions about the likelihood of these outcomes. Over time, this capability can avoid repeated calls to the Blitzkrieg module to generate the possible set of outcomes. Third, the fluents factor the search space into small dependent subspaces defined by each fluent. Between fluents, there can then be assumed to be independence or at least rare occurrence of dependence.

3.1.1. Technical Details

In this section, we'll describe the technical details of basic linear fluents. Each fluent is defined by a linear system:

$$x_{t+1} = Ax_t + By_t + Cu_t + w_t$$
$$z_t = Hx_t + v_t$$

Note that x, u, etc. are vectors of variables. x denotes output variables, i.e. state variables that are affected by this fluent. We'll often call these the modeled or relevant state variables. These variables are also necessarily input variables in this formulation; however, if we wish to preserve the notion that some output variables need not be input variables, then we simply set the proper row of matrix A to zeroes. y denotes input variables that are not affected by the output of this fluent. And u denotes control variables that can be set by the user/commander/planner. z is an observation of the output variables x, which may be observed with some noise, given by w. Similarly, the process itself is subject to noise, given by v.

w_t and v_t are random processes, which we can assume are white noise and independent:

$$p(w) \sim N(0, Q)$$
$$p(v) \sim N(0, R)$$

where Q and R are the noise covariance matrices.

This formulation has the advantage of being very close to the classical Kalman filter formulation for a linear system. Thus, if we wish to predict future states of the world, we have at our disposal an efficient algorithm for determining this information, taking into account the noisy and probabilistic nature of both actions and observations.

Extended Kalman filters could be used to model and predict non-linear systems. Or, if we don't wish to approximate using linearization, we can make use of work we have done using particle filtering methods. However, this is not the focus of the work here, so we will not discuss this in further detail. The main point is that by representing fluents as linear or linearizable systems, we have an efficient means of estimating future state, and more importantly and more precisely speaking, the distribution over future states. Instead of having to do large amounts of sampling in order to determine these distributions, we can determine the distributions in polynomial time.

3.2. Futures Graphs

A Futures Graph is a directed acyclic graph (DAG) that represents the set of possible future trajectories, given the current state and a set of possible plans. From a root node that represents the current state, each branch of the graph denotes a different potential outcome, and thus a different potential future trajectory. The set of all future trajectories is thus the set of all paths from the root to the leaves / terminal nodes of the DAG.

Each possible future trajectory is typically represented as a sequence of distributions over possible outcomes. These sequences are composed of two alternating types of nodes: fluents and state snapshots. In the previous section, we described fluents and their role in modeling states as dynamic processes. State snapshots, as their name implies, capture instantaneous descriptions of the world state, when one fluent terminates and another one begins. In both the fluents and state snapshots, at any given point in time, the state variables are maintained as distributions over their likely values.

For example, two units might be engaged in battle for some period of time. This would be modeled as a battle fluent, with equations that describe the change in unit masses over the course of the battle. When the fluent is initiated, there is no uncertainty, and the state relevant state variables (mass, position) will have a point-mass distribution over their observed value. As the fluent progresses, this distribution will increase in variance since there is typically noise in the environment (and hence in our models). The fluent would have termination conditions; in this case, the fluent terminates when 1) a unit successfully disengages from the battle, or 2) a unit is defeated and is removed from the game. In either case, the battle fluent terminates and links to a state snapshot that

describes each unit's relevant variables at this termination instant. This snapshot also captures a distribution of variables relevant to the termination, typically including time as a variable. If one unit has been defeated, then the state snapshot might capture the joint distribution over time and the victorious unit's mass at the moment of victory, as shown in Figure 2. In this example, it can be seen that battle concluding later in time benefit Unit A (the victorious unit), which ends up with probabilistically higher mass. This timing information is crucial when projecting the outcomes of future interactions with other units in the overall battle operations.

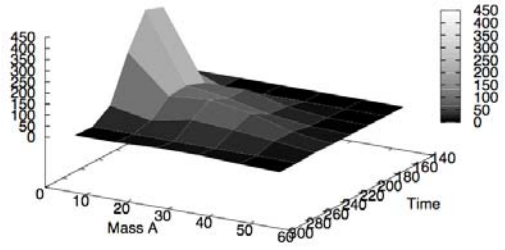


Figure 2. Joint distribution of the time and the victorious unit's mass at the moment of victory, terminating a battle fluent describing the battle between two units.

3.2.1. Branching in a Futures Graph

Branches occur in two ways in a Futures Graph. A branch can occur when the commander has multiple choices of action that he can pursue. This will be represented in the Futures Graph as a state snapshot with multiple possible fluents leading out of the state snapshot. The basic case of this type of branching occurs when the previous fluent has terminated, and the commander must choose a new course of action. Another case occurs when the commander has the option to interrupt a fluent. While state snapshots are natural decision points, where a commander might pause to take stock of the situation and decide on the next course of action, our Futures Graph also allows the commander to change their course of action mid-way through a fluent execution. This is one of the advantages of the fluent representation. Since the dynamic process inside the fluent is being explicitly modeled, it is easy to terminate a fluent at any arbitrary time, if the commander decides to pursue a different course of action. In this case, the interruption point is represented as a state snapshot, with variable state values, and two outgoing fluent branches: the first being a continuation of the fluent without interruption, and the second being the new action that is chosen instead of continuing with the current fluent.

A branch can also occur due to multiple possible outcomes of a fluent. For example, a battle fluent might terminate with one unit winning and the other unit being eliminated, or vice-versa. These two outcomes would be represented as two separate state snapshots in the Futures Graph.

3.2.2. Independent vs. Joint Fluents

So far we have focused on the computational advantages of fluents that are the result of modeling continuous processes using linear systems. There are other important sources of computational advantages as well. Fluents serve to factor the space of joint actions, allowing us to focus on independent (or nearly independent) subsets of the joint state and action space. This results in greater efficiency, since we are keeping track of fewer variables within each fluent. For example, if a unit is moving to a landmark, it is executing a movement fluent that models its change in position over time. This fluent does not need to model any of the other variables associated with this unit (assuming they are not changing), and it also does not need to model variables associated with other units. In contrast, joint fluents model interactions between units. A battle fluent models the change in masses of two or more units and the units are engaged in battle with one another.

In the Futures Graph, joint fluents can be identified by the fact that they will have two separate units as ancestors in the DAG. At some point in the history of fluents and state snapshots, two independent fluents will have combined into one state snapshot that describes state variables relevant to both units. The outgoing fluent from that state snapshot is then a joint fluent. The units could again go their separate ways if they choose to execute independent fluents at some point in the future, such as disengaging from battle and moving back to their fixed bases.

3.2.3. Updating the Futures Graph

As the battle evolves, the Futures Graph can be efficiently updated to reflect the latest battlefield observations. Since we model futures using full distributions, if we receive a trusted, reliable observation, updating simply requires taking a slice through the joint distribution, corresponding to the value of the observed variable. If more than one variable can be observed, then we can take a hyper-slice of the full joint distribution. If on the other hand, we receive a noisy observation, then we can update our model using the usual Kalman filter update.

3.2.4. Detailed Example

To illustrate the construction of these Futures Graph, we'll describe a simple, but detailed example. In this example, there are two units in the battlefield, labeled "A" and "B". There are also three general locations marked on the terrain. This is shown in Figures 1 and 2. Unit A's goal is to destroy Unit B. Unit A has three possible plans, where the first actions involve moving either directly to B, to location X, or to location Z. The search tree in Figure 2 shows several steps of the plan where the first action is to move to location X. The goal of the plan is to circle around to a southerly orientation relative to Unit B, by first moving to X, then to Y, then to B. The movements to X and Y can be modeled independently of B's actions. Joint fluents are only necessary when the two units execute actions that depend on the other unit, such as when Unit A attacks Unit B.

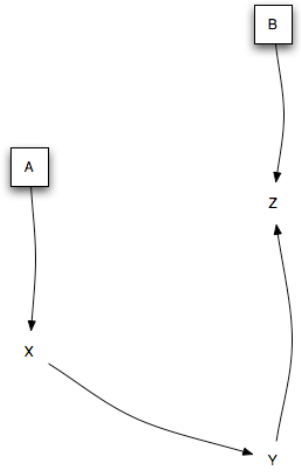


Figure 3. Unit A and B on a battlefield, with locations X, Y, and Z marked on the field.

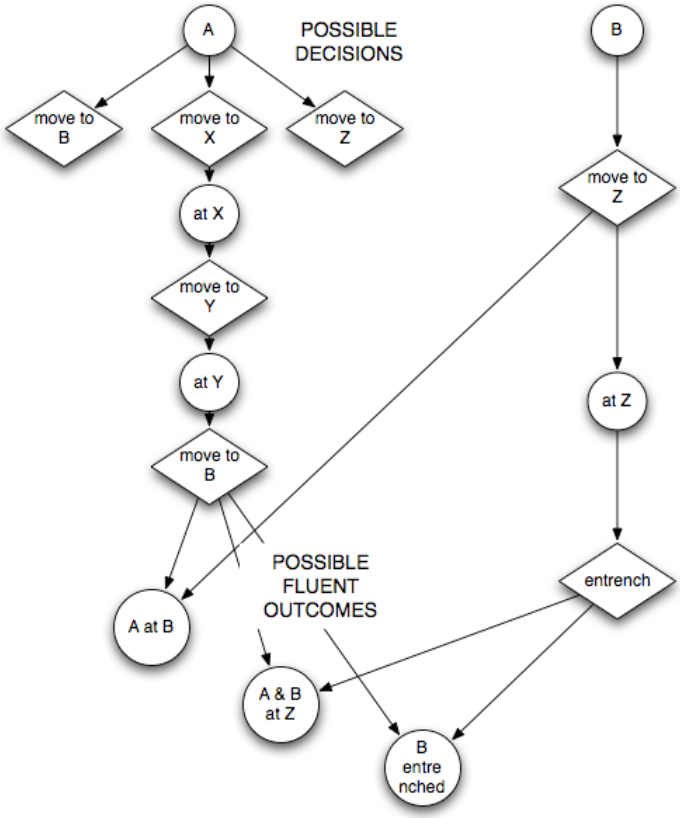
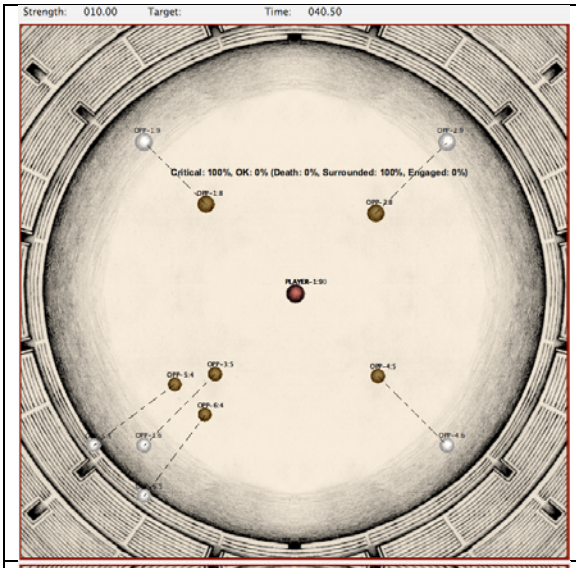
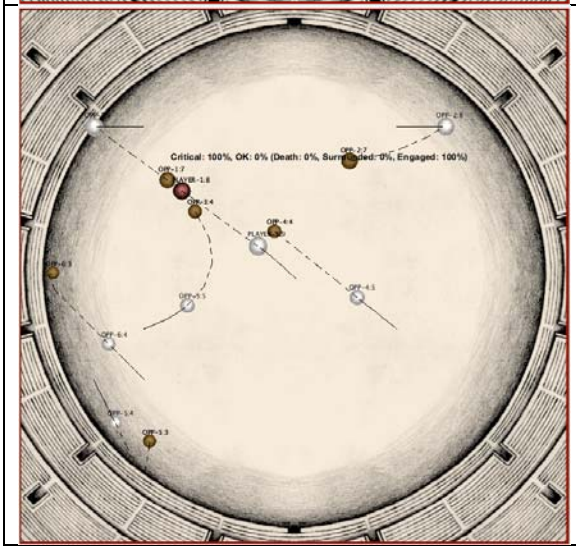
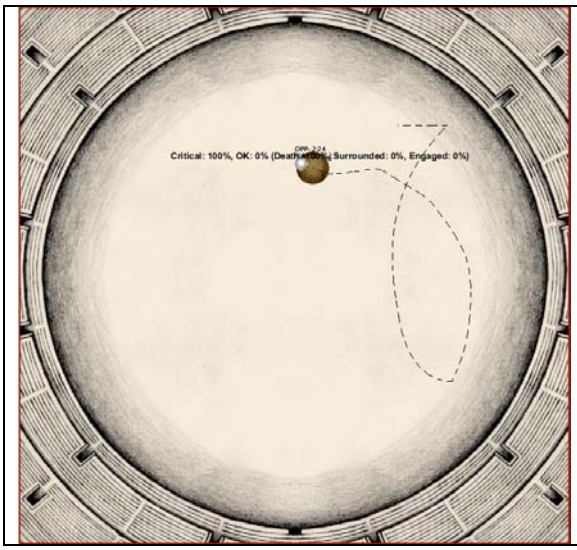


Figure 4. Search tree showing Unit A's possible decisions, and the possible outcomes of one decision path when it eventually moves to and attacks Unit B. Fluents are depicted as diamonds, and branches represent the fluent outcomes. These outcomes then lead to the next decision node, where a new fluent is executed. When two units interact, this is depicted by a joint fluent that is executed after branches from each unit's search tree lead to a single decision node. For example, in the diagram above, there are three cases where the fluent attack(A,B) will be executed next. The initial state of each of these three cases is qualitatively different.

3.3. Critical Points

Critical points occur when the commander's options are limited. A commander always tries to insure that he has reasonable alternate courses of action available, and the ACTSS system helps to insure that the commander does not end up stuck in a critical point. The ACTSS system monitors the Futures Graph for critical points, and alerts the commander to their potential occurrence before they actually become realized. In Arena War, we monitor for three types of critical points, which are described in the table below. In each of these situations, the commander has few or no good options.

	**Surrounded** Unit is ringed by multiple enemy units, rendering it unable to escape if all enemy units attack. Short-term options are limited to a constrained area between the attacking enemy units, and once battles begin, there is limited opportunity to escape since multiple units will be attacking.
	**Engaged** Unit is engaged in battle with an enemy unit, and before the battle is complete, a second enemy unit attacks. No options exist other than to finish the battle, since it is impossible to disengage when being attacked from multiple fronts.



Overwhelmed

Unit is outnumbered and will almost certainly lose the battle. The only reasonable option is to attempt to disengage, and even this option will be difficult since the enemy knows that giving chase will almost certainly result in their victory.

Figure 5. The three types of critical points monitored.

3.4. Heuristic Evaluation of Game State

While construction of the Futures Graph is fairly efficient, occasionally we will want to be able to place a rough preference over multiple possible choices of actions. To d this, we implemented various heuristics that very quickly generate quality metrics for different courses of action. The quality metrics we are most interested in are utility, likelihood, and flexibility. Utility is defined as the expected utility of pursuing a course of action. Utilities are calculated based on the goals and circumstances of the mission. Using a complete Futures Graph, we can calculate the utilities at each of the leaf nodes, and propagate these values up the tree structure to arrive at an expected utility for a particular action (branch from the current / root node). Likelihood denotes the probability with which the desired outcome will be achieved. This corresponds to the probability of a particular trajectory (or set of trajectories) through the Futures Graph. In Arena War, the desired outcome is simply the outcome where we are the victorious unit. Finally, flexibility measures the number of available options at some future point in time. In the previous section, we discussed how critical points identify the points in the future where the commander may have severely limited options.

As just described, we can compute each of these quality metrics by fully expanding the Futures Graph and using it to calculate the various metrics. However, occasionally we need faster performance, and are willing to trade-off accuracy. Thus we need a way to generate these performance metrics at intermediate nodes without necessarily fully expanding the resulting sub-trees. For Arena War, we have implemented a set of heuristics that serve this function. Eventually, we may be able to learn value functions that take the place of some of these hard-coded heuristics.

Based on these heuristic evaluations, we can rank the preferred actions and suggest these actions to the commander. In Arena War, we evaluate the quality metrics for the set of

actions corresponding to attacking the different enemy units. We do not consider non-attack movement alternatives. These attack actions are evaluated and ranked heuristically, and are displayed to the commander on the right-hand side of the game screen. The enemy units on the board are also colored according to these rankings, with deeper blue denoting preferred attack targets.

3.5. Integration with Arena War

In addition to the heuristic assistance described above, the complete ACTSS system has been integrated with the Arena War game. It helps the commander choose good courses of action that are more likely to result in a victorious outcome, and helps the commander to avoid bad courses of action that result in critical points.

To play Arena War with the help of the ACTSS system, the commander first inputs his plan of attack, as well as his expectations about the actions that his opponents will take. These plans can be input in two ways: 1) an ordered list of actions to execute, 2) a simple finite state machine that executes a particular action given the observed state. The first method simply uses a text file for input, while the second method requires writing a few lines of LISP code.

With the plans inputted, the commander can then start the game, and the ACTSS system will immediately generate a Futures Graph, and will continue to generate updated Futures Graph at fixed intervals in time while the game is played, using the game's current state as the updated initial state (root node of the Futures Graph) and information regarding the plans for the various units. The Futures Graph is displayed to the commander in a graph format; an example screenshot is shown in Figure 3. These graphs supplement the other key pieces of information conveyed to the commander: critical point notifications and COA evaluations.

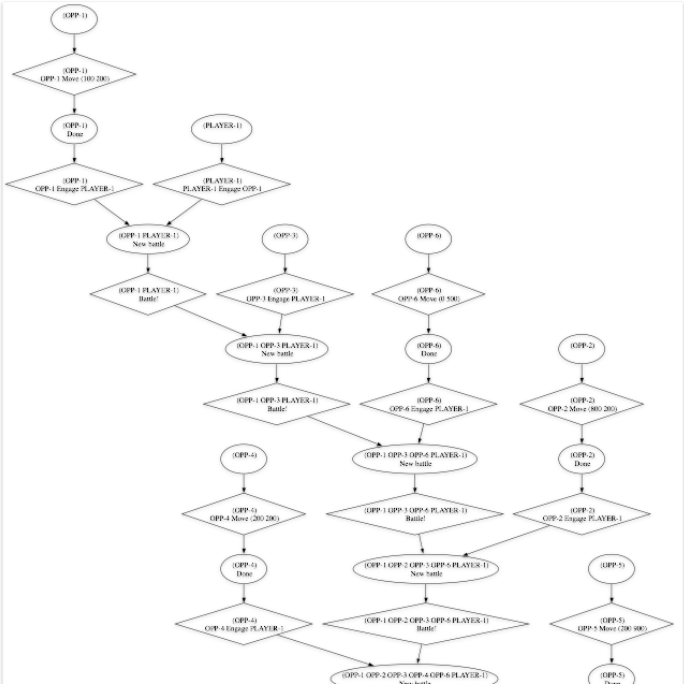


Figure 6. Screenshot of a Futures Graph presented to the commander. Diamonds represent fluents, and ovals are state snapshots.

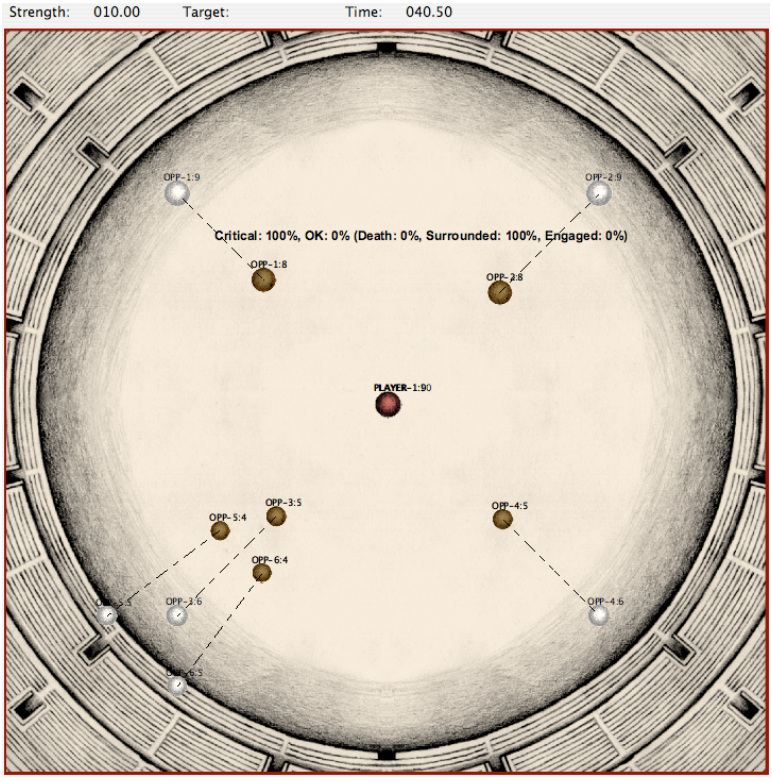


Figure 7. Critical points are displayed to the commander by visually displaying a sample from the future trajectories that result in that type of critical point.

ACTSS monitors the generated Futures Graph for critical points. If one or more are detected, then the critical points are displayed to the commander inside the actual Arena War game screen. The display "fast forwards" to an actual example critical point, as shown in Figure X. The light gray balls denote the units' current positions, and the brown balls denote their future location at the time the critical point conditions are triggered. The dotted lines indicate their expected trajectories between the current and future location. The display shows that all futures given this particular set of plans result in critical points.

The game is now paused as the commander gets a chance to re-evaluate his strategy. The commander can click anywhere on the screen to continue the game. His original plan will continue to execute until he issues a new command by either clicking on an empty location on the board (which issues a command to move to that location), or clicking on an enemy unit (which issues a command to go attack that unit). This new plan over-rides his original input plan, and will hopefully allow the commander to avoid the critical point by acting early to change the future course of events. The Futures Graph is then updated accordingly, based on the new plan and the current world state, and an updated graph (and potential critical points) are shown to the commander. This cycle repeats until the commander has either won or lost the game.

If a commander is alerted to a critical point and decides that no present alternative can prevent the future occurrence of the critical point, then he may decide to restart the game from the initial conditions, inputting a new plan of attack. This allows the commander to change his course of action even earlier, and corresponds to a real commander using the Deep Green system to test out and refine battle scenarios off-line, before commencing the actual fight.

3.6. Data and Experiments

3.6.1. Commander Success with ACTSS

We are running a small study with a few graduate students to evaluate the efficacy of the ACTSS system. The students are divided into two groups. A set of 20 game scenarios are also divided into two groups, "A" and "B". One group of students is first asked to play Arena War "A" scenarios without the help of the ACTSS system, and then they are asked to play the "B" scenarios with the benefit of the ACTSS system. The other group plays Arena War "A" scenarios with the ACTSS system first, and then plays the "B" scenarios without ACTSS.

In both cases, students playing Arena War with the help of the ACTSS system succeed more often. Of course, this is somewhat expected, since the ACTSS system allows the commander to experiment with different actions and view their consequences, without actually committing to the actions long-term. The students can thus try many possible actions until they find one that the ACTSS system deems likely to result in success.

In a real-world situation, commanders may also face tighter time constraints, and larger problem spaces. Performance will also be much more dependent on the quality of the underlying models. The linear models we used in the fluents almost perfectly match the actual game engine model that runs the Arena War game. Thus the Futures Graph predictions are highly accurate. We will also need to strive towards this kind of accuracy in modeling real-world battle scenarios. Of course, complete accuracy is unattainable, but we do not need complete accuracy. Probabilistic models will give the commander a general sense of the future trajectories possible. Also, Deep Green's key job is to alert the commander to potential critical points, and as long as the probabilistic models that we use are roughly accurate, we will be able to identify these potential critical points. Armed with this information, the commander will be able to avoid these situations and choose better courses of action.

3.6.2. Efficiency Gains

Fluents are an efficient method for modeling dynamic processes with uncertainty or noise. An alternate method would be to approximate the future state distributions via Monte Carlo sampling methods. From an initial state, a large number of samples would be drawn, and these samples would either be used to estimate the parameters for the resulting state distribution, or be kept as a large set of particles that approximate the state distribution. The size of the sample would depend on the accuracy desired.

Fluents are more efficient than Monte Carlo sampling with regards to both time and space considerations. Fluents that use linear models are particularly parsimonious with regards to space requirements, since joint Gaussian distributions can be simply stored with a vector of means and a covariance matrix. Each state snapshot in the Futures Graph can be stored in this way. The fluents themselves are represented by the linear models involved.

Projecting a fluent forward (estimating the state distribution at some future time point) can be done in time polynomial in the number of state variables and linear in the number of time steps. On the other hand, computation time for Monte Carlo sampling depends on the number of samples that need to be drawn. This number again depends on the number of relevant variables and the number of time steps, but can be extremely large if we have multiple successive fluents to consider, each with multiple dynamic variables. The samples become more sparsely distributed as we project further into the future, necessitating a much larger number of samples in order to get an accurate estimate.

Finally, fluents are always precise since they only rely on analytical solution rather than sampling methods. As long as our models are accurate, the resulting state distributions will be correct.

4. Accomplishments and Lessons Learned

We implemented a fluent-based search algorithm and demonstrated its efficacy on a domain we call "Arena War". Arena War is a free-for-all melee that requires the commander to quickly evaluate alternative plans (which unit to attack next or evade) and choose the best option. Performance depends on the ability of the commander to evaluate the eventual outcomes of each of these alternative plans using our search mechanism.

We demonstrated that a fluent-based search provides significant efficiencies over a standard joint action space search. Partly this is due to the reduction in sampling needed to generate outcome distributions in a factorized action space. We also avoid the exponential branching that could result by considering the entire joint action space at once.

The fluent-based search also achieves efficiency due to its use of linear systems to model dynamic processes. By using linear systems, we are able to project future state variable distributions in polynomial time and constant space. The assumption of normal distributions allows this projection to be solved analytically, obviating the need for computational integration to convolve and update the probability distributions as we project them forward. The resulting distribution is always precise. It is accurate as long as the models used in the fluents are accurate. In contrast, in addition to requiring an accurate model, sampling-based techniques also require a prodigious number of samples (and thus time) in order to guarantee the accuracy of the projected distribution.

Search tree complexity grows whenever there is a large number of outcomes for a fluent or a set of fluents that later interact as a merged joint fluent. Such merging can create different orderings for the completion or interruption of fluent execution. These correspond to different qualitative outcomes that we then sample separately as different branches in the search tree.

The ACTSS system also incorporates new incoming information into its search and continually updates its Futures Graph. By structuring our search using fluents and their outcome distributions, this updating can be accomplished relatively easily by taking slices of the outcome distributions and pruning the search tree accordingly. As more information comes in regarding current battlefield conditions, the search space actually becomes smaller since the uncertainty is decreasing.

Finally, we showed that we could monitor the Futures Graph for critical points, helping the commander to avoid these problematic situations. A set of heuristic quality evaluators also help the commander to quickly and roughly gauge the desirability of different action options.

4.1. Open Questions

Many interesting issues remain to be addressed, and will be the subject of future work. This section outlines some of the main areas for future work.

Expressiveness of Plans

Currently plans are input into the system by either providing an ordered list of actions, or by constructing a simple finite state machine to describe unit behavior. Eventually we will need to be able to specify more complex plans without having to write a finite state machine in LISP code. We will need actions whose execution is conditional, and we will need to be able to specify probabilistic actions. Probabilistic actions will be particularly important for allowing the commander to specify his best estimate of the opponent's behavior.

Trajectory Clustering

One important challenge for the Crystal Ball module in Deep Green is to cluster together search paths or search nodes that are similar and can thus be sampled more efficiently by considering them together. We plan to use various techniques for clustering search nodes in order to reduce the branching of the search trees. One such technique is based on our previous work on Bayesian clustering by dynamics. This will be augmented by considering the "goodness" and utility metrics we compute as we generate the search trees.

We anticipate that states might be judged similar for any of the following reasons: *Two states are similar if the values of their constituent state variables match to some degree*. The state variables include physical measurements such as position, velocity, strength; and relational variables such as distance between units; but also abstractions such as flexibility and expected utility. The fluent-based Futures Graph adds another criterion: *Two states are similar if their state variables match at approximately the same time*. A more forward-looking criterion is: *Two states are similar if they initiate similar trajectories.*} Trajectories (or snippets) might be judged similar for several reasons: *Two trajectories are similar if they go through states that are similar.* Alternatively, we might not care about how one arrived at a state, in which case, *Two trajectories are similar if they culminate in similar states*. However, even if we don't care about the states that we take to a culminating state, we might care about the dynamics of trajectories: *Two trajectories are similar if (some of) their state variables have similar dynamics*. For example, two trajectories might both ``snatch victory from the jaws of defeat," which says as much about the trajectory than the culminating state. Finally, we recognize that the Futures Graph is not simply a bundle of trajectories, it has structure, and we might wish to cluster trajectories to enhance that structure. The most important example of structure in the Futures Graph is flexibility, and an important reason to cluster trajectories is to see whether they constitute qualitatively different options or are just a bunch of variants of the same future.

Currently we do implement a kind of clustering on critical points. Critical points are clustered together based on the type of situation: Surrounded, Engaged, or Overwhelmed. Eventually we may want to be able to learn new categories of critical points, just as we learn to cluster over states or trajectories together based on the various criteria just described.

Adversarial Decision-Making

The Futures Graph is constructed by assembling the joint plans of all the battlefield units, given the set of input options/plans for each unit, provided by the commander. The commander also needs to specify the probability with which the opponent will execute each of its available options. Sometimes we may not receive this information, or the commander does not know this information, or may wish to explore different types of opponent behavior. The ACTSS system needs to get this information somehow, however, since our evaluation of utility depends on having an estimate of the probability with which the opponent executes a particular course of action.

We can compute an estimate of these probabilities by reasoning about the opponent. To perform planning in competitive domains, traditional AI often uses techniques such as mini-max pruning, which contains an inherent assumption that one side's loss is the other side's gain. However, battles are not zero-sum games like chess or checkers. Even though one side might technically win a battle or secure a location, they may have suffered such great losses during the battle that it becomes a Pyrrhic victory. In these *general sum games*, we need to apply game theoretic reasoning to consider the potential outcomes. We need to consider different outcomes based on different types of opponents: rational opponents may execute Nash equilibrium strategies, adaptive opponents may try to minimize their regret, and mini-max opponents may attempt to maximize our losses irrespective of the consequences to their own forces. Finally, we can also simply learn about the opponent's preferences over time and build up a model of the opponent based on observed behavior.

This type of reasoning will allow the commander to focus on producing rich options and broadly providing guidance on likely actions, without having to specify the probabilities of each available option. It will also allow the commander to see what kind of outcomes to expect given different assumptions about the opponent.

Plan Space Search Complexity

The problem formulation in the Deep Green project initially appears to have low search complexity since the commander is assumed to provide explicit plans (via the sketch interface) for each unit to execute. The commander can even be assumed to provide possible plans that the opponent will execute. However, two sources of uncertainty make the problem much more difficult/interesting. Both sources concern the timing of actions in the plans.

First, the stochasticity of each action in the provided plans creates uncertainty in the timing and ordering of actions later in the plan. For example, if Unit B is executing an *entrench* fluent and Unit A is executing a *move_to(B)* fluent, then we must consider at least two different qualitative outcomes depending on whether Unit B finishes entrenching before Unit A arrives and attacks Unit B.

Second, there is uncertainty involving the initiation of opponent actions. The opponent unit may delay the execution of the next action in its specified plan, whether by choice or by incompetence. In this example, the commander of Unit A might want to consider the possibility that Unit B will experience some delay before it initiating the entrench action.

Furthermore, additional complexity results from the commander's ability to sketch alternative plans, both for the friendly and enemy units. Each such alternative obviously creates more branch points. We might even allow the commander to specify default actions or sub plans that can be followed if contingencies arise. To cope with all of this uncertainty and complexity, our search methods must efficiently deal with the stochastic nature of planning and evaluation in this domain. Evaluation is particularly important since we must be able to judge the "goodness", utility, or other metrics of a given plan.

Adaptive Heuristics / Value Functions

We need to have efficient methods for evaluating performance metrics at intermediate nodes along a search path. Clearly, we could completely enumerate all searched paths to completion and compute performance metrics by performing a permanent backup to the intermediate node. However, this will not always be possible. Sometimes it may simply be too inefficient, for example, when the entropy of the outcome distributions becomes so high that such a computation would require sampling from a diverse set of outcomes and resulting subtrees.

Currently we have hard-coded a set of heuristic evaluation functions that allow us to provide the commander with quick and dirty estimates of the desirability of the available actions. In the future, these evaluation functions could be learned. This is similar to the learning of Q-functions in reinforcement learning, where actions are scored based on their expected future value given the current state.

4.2. Related Work

There are aspects of this work that are related to many existing fields. By its nature, ACTSS has many general links to the literature on search and planning. In Deep Green, planning takes on a particular flavor, since we always assume that the human is in the loop. Rather than having a completely automated planner, which is given a state representation and available action choices, in Deep Green, the human typically provides a small number of plans, and is later called upon to augment these plans with additional options as our search (projection into the future) reveals problems with the original plans.

Our fluent representation recalls some of the ideas of event-based simulation, where the simulation is updated only at specific events. However, unlike event-based simulation, fluents allow us to track and model the dynamics of the process. Fluents can thus be interrupted at any time, and it is easy to compute the resulting distribution over state variables.

The reinforcement learning literature also discusses fluent-like structures called macros or options. Macros encapsulate a lower-level process so that the planner can reason hierarchically using only the higher-level macros. However, macros do not capture dynamic change, since they still rely on the traditional model of states and instantaneous actions that transition an agent from one state to another.

SMDPs, on semi-Markov decision processes, capture some aspects of processes that take a period of time to complete. However, unlike fluents which contain a model of the process and which can thus be used to predict process completion, SMDPs simply use transitions between states that take a probabilistic amount of time.