nuWar: A prototype sketch-based strategy game

Greg Dunham, Ken Forbus, Jeffrey Usher

Qualitative Reasoning Group
Northwestern University
1890 Maple Avenue
Evanston, IL 60201 USA
{gdunham, forbus, usher}@northwestern.edu

Abstract

Today's military strategy games provide unrealistic interfaces for players to interact with their units: Commanders don't use mice and menus, they sketch. Developing strategy games currently involves grafting AI capabilities on top of a separate simulation engine, with new hand-crafted strategies for each game. We are experimenting with a novel approach for solving both problems. We started with nuSketch Battlespace, a knowledge-rich sketch understanding system developed for military users, and built a game engine, nuWar, on top of it. nuWar is a prototype two-player tactical war game which can be played either hot-seat or over a network. nuWar uses sketching as the primary way for players to express their intent to their subordinate commanders. The underlying ontology used by nuSketch Battlespace is used in both the simulation engine and in the bots which serve as subordinate commanders. We describe the architecture of nuWar, focusing on how it uses sketching, how the simulation engine is built upon the rich representational facilities of nuSketch Battlespace, and how the bots work. We discuss the tradeoffs we have found in this approach so far, and describe our plans for future work.

Introduction

Spatial reasoning is an essential tool for players of today's strategy games. To succeed, a good player must be able to think about the game world more abstractly than as a simple set of tiles. A human player can carve the space up into a variety of qualitative spatial features, devising a plan of action with respect to these features [4]. Today's games provide players with unrealistic interfaces for communicating such a plan to their units. Keypads, mice, and menus are often poor substitutes for a sketch when describing a spatial plan. In fact, military commanders have been doing this for years, sketching out battlefield plans, also known as *courses of action* (COAs) [5]. Yet it would be difficult or impossible for a general to communicate a spatially rich COA using the interface provided in most strategy games.

The lack of qualitative spatial representations in strategy games also hinders the creation of game AIs [4]. Currently, AI infrastructure is typically built on top of a separate game engine, with strategies tailored to the engine's internal workings [10]. After months of hard work, developers may be left with an AI architecture that is only useful for that one game engine implementation. The entire process must begin anew for the next game.

This paper describes nuWar, a prototype two-player tactical war game that attempts to solve these two problems. nuWar uses sketching as the primary means of user interaction with the game. It is built on top of nuSketch Battlespace [5], a system for creating COAs, which has a rich underlying ontology. After introducing nuWar, we illustrate the sketching interface that we believe makes player interaction more natural than traditional war game interfaces. We then describe the advantages and

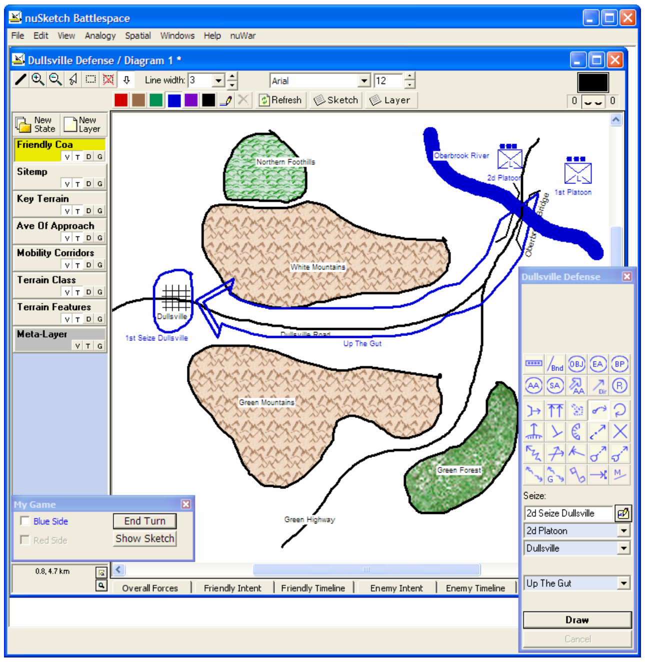

Figure 1: The Blue-side player sketches orders for a direct frontal assault on the city of Dullsville.

tradeoffs of the game's deep ontological grounding. Next we explain our use of SADL, a declarative action language, to control the actions of subordinate commander AI bots. Finally, we discuss the current state of nuWar and our plans for future work.

nuWar Overview

nuWar is a head-to-head (Red vs. Blue) war game, which can be played either as a network game or in "hotseat" mode. The interface mechanic is similar to that used by commercial games such as *Combat Mission*. There is an initial scenario, either drawn from a library of scenarios or sketched by one of the participants. Each player simultaneously looks over the situation from their side's point of view, and formulates a plan using the nuSketch Battlespace sketching interface. For example, the Blue-side player, whose sketch is seen in Figure 1, may see the pass between the mountains as the most direct path to seize the city of Dullsville. He sketches out his plan for a frontal assault on the seemingly unguarded city. The Red-side player, whose sketch is seen in Figure 2, cannot yet see any Blue units, but suspects that the enemy may try to seize the city by approaching through the pass. He sketches his orders for an ambush, laying in wait from

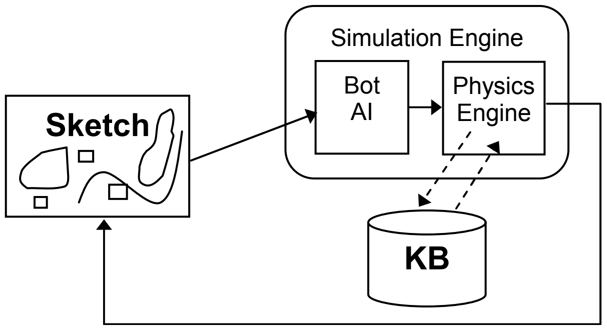

Figure 3: Architecture diagram illustrating the process used to generate each discrete "tick" of a turn movie.

well-concealed positions behind the mountains. Their plans complete, each player clicks a button indicating they are ready to execute the turn.

When both players are ready, the simulator engine runs for a predetermined amount of game-time (Figure 3), and both players get to see what happened in the form of an animated movie that uses the same graphical elements from the battlespace sketch. Figure 4 shows the end-state of the turn-movie generated from the execution of the turns described above. The underlying physics model and task execution details used to generate the turn results are described below. After the movie has been viewed, players can modify their plan, if desired, and initiate another round of game simulation.

The motivation for developing nuWar was to provide a game-based environment for knowledge capture from military experts. That is, by "watching" two experienced officers play each other in the game, we plan on building

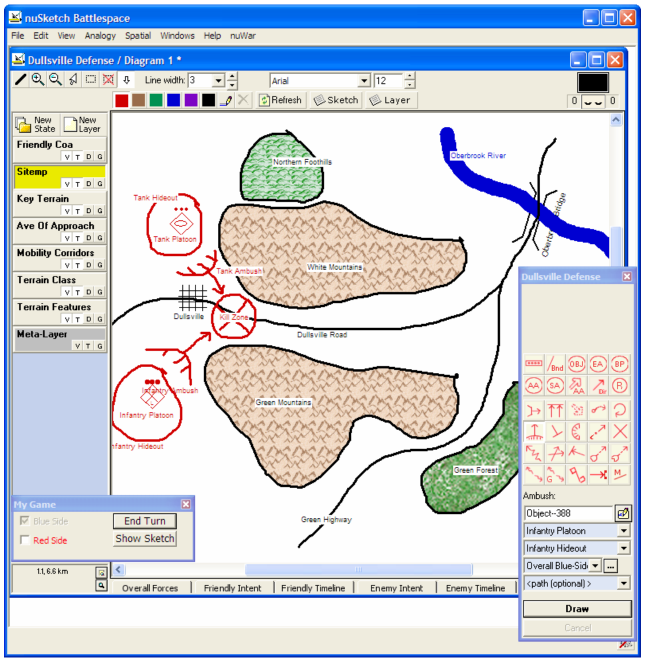

Figure 2: The Red-side player describes an ambush by sketching a course of action.

up a library of strategies and tactics. This means that the constraints on realism in the current version are tighter than might otherwise be desirable for mass-market gameplay, e.g., the large number of units, kinds of units, and their capabilities.

Sketching is a player's sole means of affecting the state of a nuWar game. A key consideration while designing the sketching interface was balancing the tension that arises

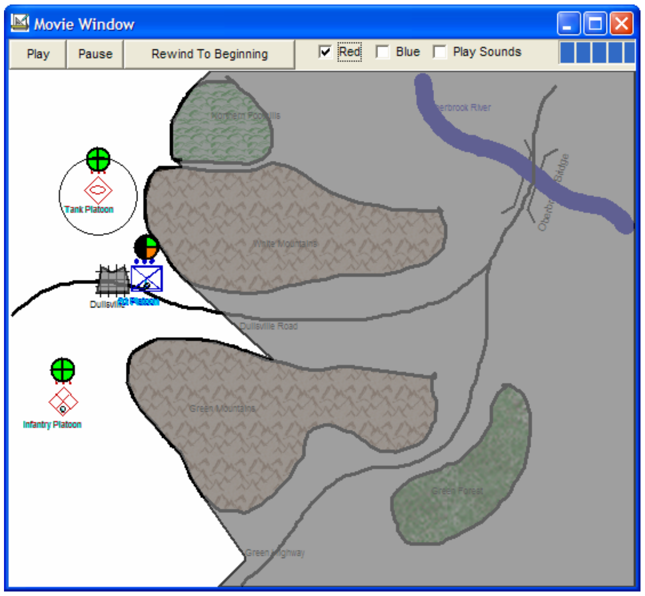

Figure 4: The results of the turn illustrated in Figure 1 and Figure 2. The Red ambush succesfully prevented Blue from seizing the city of Dullsville. Shaded areas represent regions that are not visible to Red.

between minimizing the player's input friction and maximizing the system's understanding of the player's intent so the bots can attempt to carry out what the user wanted. The next section describes how the sketching interface addresses these concerns.

Sketching as a Game Interface

Many game genres require the player to engage in spatial reasoning. Sketching can provide a more natural expression of user intent with respect to spatial concepts [6]. Sketching is a particularly good interface match for a war game because it simulates how real military commanders communicate plans, thus providing another source of immersion. For example, the graphical language of tasks that nuWar uses comes from US Army training manuals, and many of the representations and interface decisions were made in collaboration with military officers as part of the research which developed nuSketch Battlespace. While this is not a system they are using today, it may be the ancestor of what they use in the future [8].

In nuWar, the current game state is presented as a sketch of the current scenario's terrain. The locations of friendly and visible enemy units are displayed on top of the terrain layer, as are sketched graphical representations of standing military orders (tasks). The player may not modify terrain or unit locations, but may modify the tasks to suit their preferred strategy.

The nuSketch Approach To Sketching

Many sketching interfaces focus on recognition [5]. This is clearly an important approach, and has been demonstrated to provide more natural and usable interfaces to existing computer systems. Unfortunately, such systems are hampered by the poor performance of today's statistical recognition systems. Techniques like combining results across modalities (e.g., gesture and speech [2]) help, but reliable results are only obtained by carefully crafting the environment, lots of user training, and tightly circumscribing the domain of discourse. All work against immersion in a gaming environment.

The nuSketch approach is very different. We avoid recognition issues by engineering around them, providing reasonably natural, but 100% reliable, alternatives. Instead we focus on visual and conceptual understanding of the user's input, to support communication with AI systems that will reason with it.

The nuSketch Battlespace Interface

The nuSketch Battlespace (nSB) interface can be seen in overview in Figure 1 and Figure 2. Many of the elements are standard for drawing systems (e.g., widgets for pen operations, fonts, etc.) and need no further comment. The crucial aspects that make the basic interface work are *layers* to provide a functional decomposition of the elements of a sketch, *glyph bars* for specifying complex entities, *gestures* that enable glyphs to easily and robustly be drawn, and *intent dialogs* and *timelines* to express narrative information in a COA that is not easily captured by a sketch. We describe each in turn.

Layers

COA sketches are often very complex, and involve a wide range of types of entities. The use of layers in the nuSketch interface provides a means of managing this complexity. The metaphor derives from the use of acetate overlays on top of paper maps that are commonly used by military personnel. Each layer contains a specific type of information: Friendly COA describes the friendly units and their tasks, Sitemp describes the enemy (Red) units and their tasks, Terrain Features describe the geography of the situation, and the other layers describe the results of particular spatial analyses. Only one layer can be active at a time, and the glyph bar is updated to only include the types of entities which that layer is concerned with. Sketch clutter can be reduced by toggling the visibility of a layer, making it either invisible or graying it out, so that spatial boundaries are apparent but not too distracting.

Glyphs

Glyphs in nuSketch systems have two parts, *ink* and *content*. The *ink* is the time-stamped collection of ink strokes that comprise the base-level visual representation of the glyph. The *content* of the glyph is an entity in the underlying knowledge representation system that denotes the conceptual entity which the glyph refers to. Our interface uses this distinction to simplify entering glyphs by using different mechanisms for specifying the content and specifying the spatial aspects. Specifying the conceptual content of a glyph is handled via the glyph bar, while the spatial aspects are specified via gestures. We describe each in turn.

Glyph bars

Glyph bars (Figure 5) are a standard interface metaphor, but we use a system of modifiers to keep it tractable even with a very large vocabulary of symbols. The idea is to decompose symbol vocabularies into a set of distinct dimensions, which can then be dynamically composed as needed. For example, in nSB there are (conceptually) 294 distinct friendly unit symbols and 273 distinct enemy unit symbols. However, these decompose into three dimensions: the type of unit (e.g., armor, infantry, etc., 14 friendly and 13 enemy), the echelon (e.g., corps to squad, 7 in all), and strength

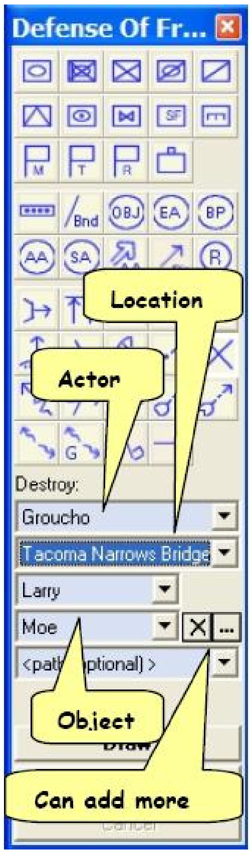

Figure 5: Glyph bar

(regular, plus, minus, or a percentage). Our glyph bar specifies these dimensions separately. Templates stored in the knowledge base for each dimension are retrieved and dynamically combined to form whatever unit symbol is needed.

Modifiers are also used to specify the parts of complex entities. Tasks, for example, have a number of roles such as the actor, the location, and so on. Widgets are added to the glyph bar whenever a glyph with parts is chosen. They include combo boxes and type-in boxes for simple choices (e.g., echelon), with drag and drop supported for richer choices also (e.g., the actor of a task). This simple system enables users to quickly and unambiguously specify roles.

Gestures

Sketching interfaces often use pen-up or time-out constraints to mark the end of a glyph, because they have to decide when to pass strokes on to a recognizer. This can be a good interface design choice for stereotyped graphical symbols. Unfortunately, many visual symbols are not stereotyped; their spatial positions and extent are a crucial part of their meaning. Examples include the position of a road, a ridgeline, or a path to be taken through complex terrain. Such glyphs are extremely common in map-based applications. Our solution is to rely instead on manual segmentation. That is, we use a Draw button that lets users indicate when they are starting to draw a glyph. There are two categories of glyphs where pen-up constraints are used to end glyphs, but in general we require the user to press the Draw button (relabeled dynamically as Finish) again to indicate when to stop considering strokes as part of the glyph.

Types of glyphs

For purposes of drawing, glyphs can be categorized according to the visual implications of their ink. There are five types of glyphs, each with a specific type of gesture needed to draw them, in nSB: location, line, region, path, and symbol. We describe each in turn. Some types of glyphs (e.g. location glyphs and line glyphs) are only available to nuWar players during scenario design, since users are restricted to giving orders once a game has started.

Location glyphs: Military units are an example of location glyphs. Their position matters, but the size at which they are drawn says nothing about their strength, real footprint on the ground, etc. Instead, this information is inferred using the knowledge base (KB).

Line glyphs: Roads and rivers are examples of line glyphs. While their width is significant, on most sketches it would be demanding too much of the user to draw their width explicitly. The gesture for drawing line glyphs is to simply draw the line.

Region glyphs: Both location and boundary are significant for region glyphs. Examples of region glyphs include terrain types (e.g., mountains, lakes, desert) and designated areas (e.g., objective areas, battle positions, engagement areas). The gesture for drawing a region glyph is to draw the outline, working around the outline in sequence.

Path glyphs: Paths differ from line glyphs in that their width is considered to be significant, and they have a designated start and end. Path glyphs are drawn with two strokes. The first stroke is the medial axis–it can be as convoluted as necessary, and even self-intersecting, but it must be drawn as one stroke. The second stroke is the transverse axis, specifying the width of the path. Based on this information, nSB uses a constraint-based drawing routine to generate the appropriate path symbol, according to the type of path.

Symbolic glyphs: Symbolic glyphs do not have any particular spatial consequences deriving from their ink. Military tasks are an example. In some cases there are spatial implications intended by the person drawing it that would be missed with this interpretation, i.e., a defend task is often drawn around the place being defended. Unfortunately, these implications are not understood by the system. Instead, players use the glyph bar to specify participants in a task. Players do slightly more work this way, but the payoff is no misunderstandings.

Entering Other Kinds Of COA Information

In the military, courses of action are generally specified through a combination of a sketch and a COA statement, a structured natural language narrative that expresses the intent for each task, sequencing, and other aspects which are hard to convey in the sketch. The *intent dialog* enables the purpose of each task to be expressed. Intent is important in military tasks because it tells those doing it why you want it done. If the prescribed task's execution goes awry, subordinate commanders are empowered with the information to adapt their actions in line with the original purpose.

The intent dialog contains a template for each task. A task is basically an order (i.e., "seize Dullsville"), specified by the mechanisms outlined earlier. The purpose is conveyed by filling out a template: <*task*> in order to <*modal*> <*agent*> from/to <*action*> <*object*>. <*modal*> is one of enable, prevent, or maintain. Agent is a unit, multiple units, or a side. <*action*> is drawn from a set of actions or states (e.g., "controlling") and <*object*> is the thing that <*action*> would affect. Thus "1st Platoon seize Dullsville in order to prevent Red from controlling Dullsville" is an example of the kind of statement that can be generated, by using pulldowns or drag and drop with the intent dialog. The output from the dialog is a formal representation of this intent that can be understood by the rest of the system.

Timelines are Gantt-chart dialogues that enable temporal constraints to be stated between tasks. Constraints that cross sides cannot be stated, since typically one is not privy to the other side's planned tasks. One task can be constrained to start or end relative to the start or end of another task, or at some absolute point in time. Estimates of durations can also be expressed.

Common to each of these sketching interface elements is an underlying entity in the knowledge representation system. The next section describes the ontology, and the architectural benefits that arise from employing it as a cornerstone of the system.

Grounding a game engine in a rich ontology

The nuWar physics engine, the subordinate bot AI, and the nuSketch Battlespace sketching application all exploit a large, common, knowledge base to support their core functionality. This arrangement simplified game engine development because the framework for representing specific knowledge about units, weapons, and terrain already existed in the KB. Changing fundamental game properties is also simplified. For example, adding a new type of unit or weapon does not require changes to the game code. Instead, a change to the KB makes the same information immediately available to the interface and the bots.

The knowledge base used by nuWar is a subset of the contents of Cycorp's Cyc KB [7] with some limited custom extensions. The KB contains specialized military concepts as well as a wide variety of more general common sense facts.

The architecture of the nuWar simulation engine is, in many ways, not novel. The system performs a standard incremental-time simulation, using Lanchester equations [9] to calculate weapon effects. Ammunition and fuel consumption are modeled, along with attrition. What is different about the nuWar approach is (a) how the entities and their properties are specified, and (b) how the spatial components of the calculations are done.

The entities in a nuWar sketch are two-dimensional spatial representations associated with conceptual facts in the underlying ontology. Because scenarios are composed of sketched input, spatial entities are coarse representations of terrain properties rather than tiles. Ink is used to provide the boundaries of entities in the sketch, which are required to perform spatial reasoning about the sketch map. nuSketch combines this information with knowledge from the KB about units and terrain to calculate Voronoi diagrams [3] and polygon operations used in determining a variety of spatial relationships, including visibility and fog of war. For example, visibility is computed by first deriving a set of obstacles based on the properties of the entities around a unit (looked up from the KB), then using line-of-sight computations with distance clipping to derive the region that a unit can see. The union of these polygons for a side indicates what that side can see, and the complement is that which is obscured by the fog of war. Mobility rates are computed using terrain knowledge provided by the ontology, using coarse approximations derived from Army manuals and expert input. These spatial reasoning abilities provide new capabilities to bots: Both position-finding and path-finding [6] can take as argument constraints like "avoid being seen by the other side", for instance.

Employing a structured ontology as the foundation of a game is not without cost. The developer must have access to a KB with an adequate number of facts relevant to the game's domain. Tools may also be required to interface with the KB. nuWar was able to minimize these costs by making use of the well-established Cyc KB's contents and a pre-existing reasoning engine, built into nSB, which can reason with that content.

nuWar's use of formal knowledge representation is not limited to the underlying world model. The next section describes nuWar's use of a task description language to guide the player's subordinate commander bots in carrying out sketched tasks.

Declarative guidance for bots

Each military unit in a nuWar game has a bot acting as subordinate commander to carry out the player's orders. There are 19 different types of tasks that a player might assign to a unit, and so the bots must be capable of performing 19 tasks of varying complexity. The low-level, in-game actions available to units in nuWar are generally very limited. A unit can remain in place, move to a location, follow a path, assume a different posture, attack another unit or tactical target, and a few others. The higher-level tasks available to nuWar players are generally more complex, but can all be composed using combinations of these simple actions. The behavior for the few primitive actions is hard-coded into the physics engine, meaning that any changes to the fundamental building blocks require fairly intensive code modifications. Since the higher-level tasks can all be specified using this small vocabulary, we were then able to quickly describe them using declarative task representations.

While supporting declarative task representations

```
(every Seize-MilitaryTask has
  (sub-events
   (a Assume-Offensive-Posture called "assume-offensive-posture" with
     (next-event (((the sub-events of Self) called "move-along-path"))))
   (a Move-Along-Path called "move-along-path" with
     (next-event
       (if (lisp (distance-to-end-of-path (nuwar-unit Bot)))
         then ((the sub-events of Self) called "move-along-path"))
       (if (lisp (not (distance-to-end-of-path (nuwar-unit Bot))))
         then ((the sub-events of Self) called "occupy-position"))))
   (a Occupy-Region-Center called "occupy-position" with
     (next-event
       (if (lisp (distance-to-target-region (nuwar-unit Bot)))
         then ((the sub-events of Self) called "occupy-position"))
       (((the sub-events of Self) called "attack-target"))
         (if (lisp (not (distance-to-target-region (nuwar-unit Bot))))
           then ((the sub-events of Self) called "wait-for-new-orders"))))
  ...
```

Figure 6: A partial SADL description of the "seize" task, which calls for a unit to clear a designated area and obtain control of it.

required additional specialized code early in the development cycle, doing so offered several advantages. New tasks could be introduced and incrementally refined without disturbing the code base. Task implementation became a formal representation challenge instead of a low-level coding challenge, and was more accessible to domain experts who were not as well versed in the game engine's internals. Additionally, the representational choice significantly lowers the barrier to future reasoning about learned strategies—structural comparisons of the task descriptions could lead to strategic experimentation with "similar" tasks.

SADL

The tasks available in nuWar are described using a slightly modified version of the SHAKEN Action Description Language (SADL) [1]. We chose SADL because of its relatively simple (yet still powerful) syntax, its clearly documented feature set, and its compatibility with software tools used in-house. A SADL process description consists of a set of sub-events that may optionally specify the next event to be executed. We extended the language to allow for conditional branching based on Lisp callouts to sensor data provided by the physics engine. The physics engine, in turn, draws on the KB to infer the result for the bot. Another deviation from the SADL specification is ordered execution of sub-events, which allows for simpler authoring of task descriptions.

Figure 6 is a portion of the SADL task description for the `seize` task. The main sub-events of the seize task seen in the diagram are the primitive actions represented by `Assume-Offensive-Posture`, `Move-Along-Path`, and `Occupy-Region-Center`. After assuming an offensive posture, the bot will begin executing the `move-along-path` sub-event. The bot will continue to follow the task's specified path until the end is reached. Task execution will then shift to the `occupy-position` sub-event, which attempts to occupy the middle of the target region. If successful, the bot will remain in place and wait for new orders.

Discussion & Future Work

nuWar is currently in alpha test, mostly played by developers, other members of the research group, and a few select others. The core functionality and interface are generally working, but the user experience has not yet been polished. Although additional refinement is needed before nuWar could be considered a final product, we believe it serves as proof-of-concept for a sketch-based strategy game built on top of a rich ontology.

As noted above, the purpose of nuWar development was to provide a game-based knowledge capture environment that would engage expert military officers. However, we believe that these ideas are applicable to a wide variety of strategy games. Sketching can help provide an engaging interface in strategy games due to the strong spatial components of the domain and the resulting player strategies. Building games within a knowledge-rich framework can simplify engine infrastructure and allow for greater flexibility during the game development cycle.

We believe that the nuWar architecture will be useful for experimenting with strategic knowledge capture to support the creation of more humanlike strategy AIs for games. If generalized strategic knowledge could be captured and applied to similar knowledge-rich domains, AI development time could be cut drastically for new games.

Acknowledgements

This research was supported by the DARPA Rapid Knowledge Formation (RKF) Program.

We thank Tom Ouyang, Dan Schmidt, and Vern Chapman for invaluable programming and modeling assistance.

References

1. Blythe, J., SADL: Shaken Action Description Language, http://www.isi.edu/expect/rkf/sadl/
2. Cohen, P. Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L., and Clow, J. 1997. QuickSet: Multimodal interaction for distributed applications. *Proceedings of the Fifth Annual International Multimodal Conference*. Seattle, WA.
3. Edwards, G. and Moulin, B. 1998. Toward the simulation of spatial mental images using the Voronoi model. In Oliver, P. and Gapp, K.P. (Eds) 1998. *Representation and Processing of Spatial Expressions*. LEA Press.
4. Forbus, K.D., Mahoney, J.V., Dill, K. 2002. How Qualitative Spatial Reasoning Can Improve Strategy Game AIs. *IEEE Intelligent Systems*, Vol. 17, Issue 4, July-August 2002, 25-30.
5. Forbus, K., Usher, J., and Chapman, V. 2003. Sketching for Military Course of Action Diagrams. *IUI'03*, January 12-15, 2003, Miami, Florida.
6. Forbus, K., Usher, J., and Chapman, V. 2003. Qualitative Spatial Reasoning About Sketch Maps. *Proceedings of the Fifteenth Annual Conference on Innovative Applications of Artificial Intelligence*, Acapulco, Mexico.
7. Lenat, D. B. (1995). CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM* 38, 33–38.
8. Rasch, R., Kott, Al, and Forbus, K. 2002. AI on the Battlefield: An experimental exploration. *Proceedings of the 14th Innovative Applications of Artificial Intelligence Conference*, July, Edmonton, Canada.
9. Taylor, James G. 1983. Lanchester Models of Warfare, *Ketron Inc.*, Arlington, VA
10. Tozour, P. 2002. The Evolution of Game AI. *AI Game Programming Wisdom*, Charles River Media, 2002.