Proceedings of the Ninth Annual Conference on Advances in Cognitive Systems Submitted 9/2021; published 11/2021

Operationalizing Tactical Representations

Thomas R. Hinrichs T-HINRICHS@NORTHWESTERN.EDU
Kenneth D. Forbus FORBUS@NORTHWESTERN.EDU
Department of Computer Science, Northwestern University, Evanston, IL 60208 USA

Abstract

Tactics comprise a middle ground of abstraction between primitive actions and strategies. In this paper, we describe how an autonomous agent can use relatively general tactics as background knowledge to achieve complex behaviors in a manner that facilitates experimentation and learning. We describe our representation of tactics, their role in capturing intent in a course of action (COA), the incorporation of qualitative representations in tactics, and their benefits and challenges for implementing control strategies.

1. Introduction

When learning and acting in a dynamic environment, an agent can benefit from having general background knowledge about how to achieve goals, independent of the particular situation or environment. We think of this content as *tactics*, which occupies the middle ground between primitive actions and strategy. If strategy describes how goals are decomposed and tradeoffs are resolved, then tactics describe how actions combine to influence quantities and achieve goals.

We claim that there is value in representing tactics declaratively in a domain-neutral way, as opposed to constructing and using hierarchical task networks (HTNs), reasoning solely from cases, or planning from scratch. The benefits can be categorized as: *penetrability* – the ability to reason about a tactic, its requirements, resources and dependencies; *expressiveness* – the ability to express more complex behaviors with flexible sequencing constraints, hierarchical structure, multiple coordinated agents, and event triggers; *learnability* – the ability to decompose a tactic into individually learnable decisions; and *transferability* – the ability to operationalize a tactic in a new domain rather than reinvent it from scratch.

Such a flexible representation places demands on an agent's control mechanism. While the Companion cognitive architecture [Forbus & Hinrichs, 2006] provides a basic task agenda to support durative actions with complex sequencing constraints, to fully instantiate and execute tactics requires new capabilities for translating between general tactics and domain-specific actions, opportunistic instantiation and dynamic propagation of role bindings. We refer to this process as *operationalizing* tactics in a domain, a term introduced in (Mostow, 1981)

Our domain for exploring these ideas is Freeciv,[1] an open-source strategy game based on Sid Meier's Civilization II. Although we have been working in this domain for over a decade (Hinrichs & Forbus, 2007;2011;2012), we have not previously focused on combat tactics. These are especially challenging because they depend on timing, event triggers, and execution-time decisions

[1] http://freeciv.wikia.com

in ways that are difficult to infer from a qualitative model or to generalize from game traces because they would require an inordinate number of examples with indefinite scope. Consequently, we chose to instantiate a small number of general, well-known tactics that could be broadly applicable outside of Freeciv. This aligns with our long-standing desire for a system that can learn from historical precedent, as well as from game-playing experience.

This paper describes the benefits of a declarative representation of tactics both for reasoning about intentions and for enhancing the flexibility of behavior and control strategy. In the next section, we provide some background on Freeciv and the Companion architecture. Section 3 describes our representation for tactics. We discuss a vocabulary of tactic types that go beyond the domain of military actions and explain how the representation incorporates concepts from qualitative modeling. In Section 4, we present the mechanisms used to operationalize tactics, through translation into domain-specific actions and events. Section 5 presents an informal assessment of how well this works in the domain of Freeciv, Section 6 reviews related work on tactics representations, and Section 7 discusses the idea more generally, along with suggestions for future work.

2. Recap of Freeciv and the Companion Architecture

Freeciv is an open-source strategy game in which players build and manage civilizations and ultimately win by conquering or destroying other civilizations (There are other ways to win, but they are not relevant for our purposes). The features of the game that are important here are that a player may have many different units to control (a unit represents either an individual like an explorer, or a group actor such as a Legion), and that a turn in the game is really a phase in which all players can move their units simultaneously until they run out of movement points, a parameter based on the unit's type. Some units can move multiple tiles in a turn, while others can only move one. The game itself does not actually define primitive actions – there are gestures whose effects are context sensitive. For example, in some contexts, moving a unit in a particular direction will change its location, but if an enemy unit is occupying the destination tile, the effect is to attack that unit. Building a player for the game requires deciding which conceptual actions are distinct primitives and implementing those primitives in code.

The Companion cognitive architecture provides, among other things, extensive symbolic reasoning capabilities, a knowledge base with a broad ontology of concepts and relations, a suite of mechanisms that perform and build on symbolic analogy, Hierarchical Task Network (HTN) planning and execution, an agenda mechanism for scheduling HTN tasks based on priority and precedence constraints, and an event system that can schedule and run tasks through asynchronous callbacks. A Companion is able to play Freeciv by extending the set of predicates that can be queried to reveal the state of the game simulation, and the set of primitive actions it can execute to drive the game. The tactical planning described in this paper is not dependent on these Freeciv-specific predicates and primitives, but instead constructs and caches a mapping in order to translate between generic and domain-specific entities and actions on the fly.

It may be helpful to clarify some terminology as used in this paper. An *action* is a primitive executable command that is issued by the player-agent, in this case to the Freeciv game engine. An *event*, on the other hand, is a reified instance of an Event in the ontology. An event need not be initiated by the player, but may instead represent an action by some other player, or some other detectable state change. All actions are kinds of events, but the converse is not true.

Task is, unfortunately, an overloaded word. An *HTN task* refers to a complex activity that can be reduced to a sequence of primitives by the planner. An *agenda task* encapsulates an HTN task or action sequence with additional precedence constraints, a priority, subtask links such that it can be scheduled on a Companion agenda, and a current status, such as pending, active, blocked, or failed. These properties permit reflective reasoning so that an agent can defer or reschedule tasks or invoke failure handlers as needed.

Goals are desired states to achieve or directions in which to drive quantity fluents, for example, maximizing gold in the treasury. In the Companion architecture, type-level goals exist in a hierarchical network in which multiple goals may trade off with each other. Each goal type maintains an activation level, which is the means by which tradeoffs are addressed. *Strategies* in a companion are processes that influence the activation levels of goal types to resolve tradeoffs.

An *intent* describes a commitment to a future action or activity to address a goal. Lastly, a *tactic* is a reified knowledge-level procedure defined in terms of actions and events that addresses a particular goal or goal type.

3. Tactical Representation

Although tactics are commonly thought of in terms of military maneuvers, we intend a much broader interpretation here. We associate tactics with *systems*, so that there may be different types of tactics, including:

- Legal tactics (lawsuit, incorporating, plausible deniability);
- Business tactics (advertising, mergers, divestment, monopolizing);
- Transportation tactics (building road networks, bridges, ports and airports);
- Social tactics (flattery, blackmail, revenge).

Within military tactics, there are numerous offensive and defensive types. The broad scope of what counts as a tactic means that their representation should comprise general predicates and entities. We take as a starting point parts the ontology for events from OpenCyc.[2]

One of the main features of Cyc's event representation is that it is Neo-Davidsonian (essentially frames with slots), rather than a positional representation with an action predicate and arguments. This provides the ability to refer to incompletely specified tactics as partial representations of intent, and makes it possible to make explicit, independent decisions to fill open roles. Also, given a newly-created entity, such as a military unit in a game, the intent representation simplifies searching for entity assignments without repeatedly assigning the same task to multiple entities. Another feature of the Davidsonian representation is that it provides explicit event structure. Consider that a tactic is fundamentally an event. This may persist over a long time, coordinate multiple actors, proceed through multiple phases or states, suspend activity until some condition obtains, and overlap with other tactics. This is possible because the subevents of a tactic have names that allow for arbitrarily complex relationships among them.

[2] www.cyc.com

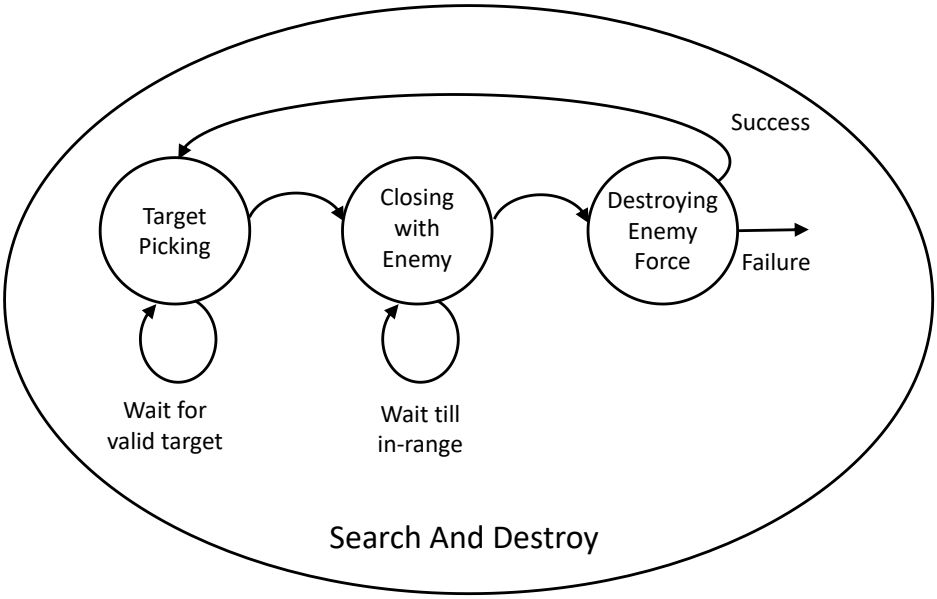

Figure 1. SearchAndDestroy as a state machine.

The actual individual behaviors described here are not terribly complex. Consider Figure 1, which shows a simplified state diagram for the SearchAndDestroy tactic. The overall tactic has three subevents that are roughly sequential: picking a target, closing with the target, and attacking the target, which results in the destruction of either the defender or the attacker. Upon success, it loops back to pick another target.

Despite its simplicity, this suggests some challenges for pure plan-based behavior: It entails waiting for a condition and it loops indefinitely. Representing SearchAndDestroy as a Neo-Davidsonian event with subevents makes this control more declarative and explicit, although not quite as simple as the state diagram suggests. Table 1 shows most of the types of statements that encode SearchAndDestroy. This is actually a specification for *instantiating* an individual SearchAndDestroy tactic. Any particular instance would have uniquely named events and participants. We describe the different kinds of statements in turn.

First, the type of the tactic is defined via `isa`s and `genls` statements. This is useful in searching for tactics to address active goals. For example, SearchAndDestroy is known to be an offensive tactic. If the game is in a phase where conquering opponents is important, then the agent will search for offensive tactics whose effects most closely match its offensive goals. Next, the `participantType` statements define named roles for entities in the tactic and specifies a type constraint to restrict their bindings. The importance of naming participants rather than specifying positional arguments is that one can represent the intent to apply a tactic, even it if is incompletely specified. The hierarchical structure of subevents is defined by `properSubEventTypes`, or `candidateProperSubEventTypes` for optional events. The `startsAfterEndingOfInSituationType` statements specify temporal constraints between subevents in the context of the parent tactic. This lets subevent types be reused in different tactics with different sequencing restrictions. Once instantiated, these temporal constraints can be directly interpreted by the Companion agenda. The next type of information is inheritance

Table 1. Type-level representation of Search And Destroy

```
(isa SearchAndDestroy TacticType)
(isa SearchAndDestroy DurativeEventType)
(comment SearchAndDestroy "A SearchAndDestroy mission is the extended event
of looking for targets, maneuvering, closing on, and destroying them.")
;;; Connect to rest of ontology
(genls SearchAndDestroy MilitaryTactic)
(genls SearchAndDestroy OffensiveTactic)
(genls SearchAndDestroy PurposefulAction)
;;; Participants in this tactic
(participantType SearchAndDestroy performedBy MilitaryAgent)
(participantType SearchAndDestroy eventOccursAt GeographicalRegion)
(associatedRoleList SearchAndDestroy (TheList performedBy eventOccursAt))
;;; Types of subevents – there can be many instances of these
(properSubEventTypes SearchAndDestroy TargetPicking)
(candidateProperSubSituationTypes SearchAndDestroy
              MilitaryManeuver-Offensive)   ; getting into position
;;; Sometimes one waits for best advantage, surprise, etc.
(candidateProperSubSituationTypes SearchAndDestroy Waiting)
(properSubEventTypes SearchAndDestroy ClosingWithEnemy)
(properSubEventTypes SearchAndDestroy DestroyingAnEnemyForce)
(focalProperSubsituationTypes SearchAndDestroy DestroyingAnEnemyForce)
;;; Repeat for more targets
(properSubEventTypes SearchAndDestroy RescheduleParentTask)
;;; Start of temporal constraints among these tasks
(startsAfterEndOfInSituationType SearchAndDestroy
     MilitaryManeuver-Offensive TargetPicking)
(startsAfterEndOfInSituationType
     SearchAndDestroy Waiting MilitaryManeuver-Offensive)
[…]
;;; Find new enemy to destroy after the current one is finished off
(startsAfterEndOfInSituationType SearchAndDestroy RescheduleParentTask
     DestroyingAnEnemyForce)
[…]
;;; Propagate the target to subtasks
(inheritRolePlayers SearchAndDestroy TargetPicking DestroyingAnEnemyForce
     intendedMaleficiary intendedMaleficiary)
[…]
;;; Describe the goal this tactic addresses
(tacticAchieves SearchAndDestroy
  (MinimizeFn
    ((MeasurableQuantityFn cardinalityOf)
     (CollectionSubsetFn Agent-NonGeographical
      (TheSetOf ?unit
       (and (opponentsInConflict
              (IndexicalFn currentRole) ?opponent
              (IndexicalFn executionContext))
            (sovereignAllegianceOfOrg ?unit ?opponent)
            (isa ?unit Agent-NonGeographical)))))))
```

constraints between roles in different subevents, specified by `inheritRolePlayers`. Because roles in subevents may have different names, the explicit mapping lets bindings be propagated at

instantiation time, and in some cases at execution time (e.g., for decisions such as the `intendedMaleficiary`, which is the target). The final information in the table is a `tacticAchieves` statement that describes the goal this tactic aims to address. In this case, the goal is to minimize the number of enemy units. The vocabulary in the goal description is generic, rather than specific to Freeciv. Mapping from one to the other is the main challenge for operationalizing tactics.

Note that this summary leaves out the specifications of the constituent subevents, which are each represented as in Table 1. Every subevent has its own event type (which need not be a tactic) as well as participant type definitions. This summary also omits any executable HTN plans that may be associated with a tactic. Reducing a tactic to executable actions is a key part of the operationalization process, which we describe in the next section.

4. Operationalizing Tactics

To make a tactic representation drive behavior requires that it be 1) instantiated in the vocabulary of the current domain, 2) reduced to executable plans and actions, and 3) scheduled in a way that allows behavior to be driven by these action sequences and by asynchronous external events. The element of the Companion architecture that supports this is called the *tactical planner*. This maintains a distinct Course Of Action microtheory (COA) as a kind of blackboard on which to record the agent's intentions as partially instantiated tactics. When a tactical event's roles are completely instantiated, it is scheduled for execution on the Companion agenda.

4.1 Opportunistic Instantiation

In a domain where entities can be created or destroyed dynamically, it is important that tactics be instantiated opportunistically, as resources become available, rather than in a fixed order in advance. The instantiation process involves proposing a tactic type to pursue, identifying candidate participant instances to fill its roles, and recursively instantiating its subevents. The tactical planner does this automatically in a domain-neutral way by using the architecture's general event-handling mechanisms to insulate it from domain code, and by creating and caching mappings between domain entity and event types and their general parent concepts. Thus, proposing a tactic type is triggered by a `GoalActivationEvent` that is fired by the architecture, provided the domain code updates it whenever entities are created or destroyed. The search for an appropriate tactic matches the domain goal against the tactical effect (from the `tacticAchieves` relations) using the cached vocabulary mapping to translate.

Given a tactic type to instantiate, the tactical planner examines the participant types and looks for available instances, such as idle military units, cities, systems, and regions. When instances are available, they are assigned to the particular participant role of the tactic in the COA. If all the participants in a tactic are bound, then it recursively instantiates the subevents and propagates participant bindings to the subevents, such as the performer of the tactic.

This selection of concrete participants to ground a tactic instance is a parametric decision that must be learned in the specific domain (e.g., Freeciv). Initially, the tactical planner selects participants arbitrarily from among entities that meet the type constraints. It progressively refines its selection criteria as it gathers decision cases that succeed or fail. Although this particular learning mechanism is beyond the scope of this paper, the relevant point is that the tactical representation replaces an intractable problem of invention with a far simpler problem of learning to make particular kinds of decisions.

4.2 Reduction to Executable Plans and Primitives

To actually apply a tactic in a domain requires resolving down to plans and primitives that can be executed in the architecture. For a domain like Freeciv, there is a simulated external environment and a set of primitive actions to manipulate it. Without such a domain simulator, the architecture only provides primitives to manipulate facts in memory, support input/output, propagate bindings, and generally drive the architecture. Although this might suffice for pure qualitative envisionment or decision-making tasks, driving an external simulator requires associating the leaf events with domain-specific actions or plans. To this end, we construct a domain-specific equivalence map between events and domain actions, and provide a mapping between named participant roles in the event and positional arguments of the plans and actions.

For example, SearchAndDestroy bottoms out in a task to DestroyEnemyForces. The tactical planner maps this directly to the domain action doAttack, which has associated code that tells the game that a particular unit attacks another. Part of the tactic representation is the mapping statement:

```
(associatedRoleList DestroyingAnEnemyForce
            (TheList performedBy intendedMaleficiary))
```

The tactical planner operationalizes the action by substituting the individual units bound to the performedBy and intendedMaleficiary roles into the first and second arguments in the action.

4.3 Translating and Scheduling Agenda Tasks

When a tactic's participant roles are fully instantiated and an action or HTN method is specified, the tactical planner packages it for execution on the Companion agenda. It creates a data structure that contains the name of the task, the action or action sequence, the parent task and subtask relations, and any sequencing constraints. The agenda is responsible for tracking the status of tasks, honoring their scheduling constraints, and executing their actions.

The main purpose of an agenda is to allow behavior to loop indefinitely, as opposed to a plan, which typically finishes at some point. At some level, looping is as simple as executing actions that reschedule the task. In practice, there may be local state that must be reset and propagated. For example, subtasks to attack a particular unit or build a particular road must be cleared once they execute so that new targets or roads can be decided. This clearing and propagating is also handled through executable action primitives. Smaller scale looping can also result from tasks that do not run to completion, but must suspend themselves to interleave execution with other tasks. This is especially true in a turn-based game like Freeciv, where actions may deplete move points and need to be resumed in the next turn. In this case, the task is not restarted, but the unexecuted portion is deferred until the next turn.

In addition to looping and waiting, some tasks may involve decision-making. For example, TargetPicking is an event whose effect is to make a decision at execution time, rather than to change the state of the simulated game. This decision must be propagated to subsequent events that close with and attack the target. Those events cannot be completely instantiated on the COA until the target is known, so the decision task invokes an action to reinvoke the tactical planner to propagate the decision and finish instantiating the subtasks.

Some tasks, such as road building, can fail either because the worker building the road is killed or simply because an enemy unit stands in the way. In the latter case, the link fails, but the overall tactic should not. We address this issue by having explicit exception-handling tasks. The

`startsAfterFailureOfInSituationType` predicate associates an exception handler with a defeasible task so that the tactic can continue even when individual links fail.

5. Assessment

We have focused initially on military and economic tasks to prove out these ideas, and currently have three tactics fully implemented: search-and-destroy, defending a position, and building path networks (see Figures 2 and 3). We chose to start with these because they address known weaknesses in our agent's behavior:

- Offensive actions were weak because despite a goal to minimize the number of enemy units, the idea of seeking out and pursuing enemies and maintaining that role could not be synthesized from the qualitative model.
- Defensive behavior suffered because even though the qualitative model led to having military defenders in cities, the lack of an explicit role or intent meant the defenders would leave to pursue any threat that passed by.
- There was no investment in distributed infrastructure because the gap between high-level system goals and primitive actions was too great. Individual roads would get built because they contributed to tax revenue, but no transportation network was ever constructed.

Rather than attempt to reinvent these tactics from scratch or build them in as code in the player, the tactics provide larger building blocks for learning and offer more flexible and reusable behaviors for better game play. Test runs demonstrated that the implemented tactics work as intended. Units now pursue mobile adversaries and attack them when they are in range (sometimes suicidally). Defenders stay in their cities and only attack adversaries when they can do so without leaving their city. Workers build networks of roads between cities and recover when their way is blocked. Notice that when a tactic fails it is usually due to a particular decision that has gone wrong – picking the wrong target, choosing to attack when it should not, or building a road in a vulnerable location. Current work focuses on learning to make these individual decisions better to optimize the tactic in the domain.

Another way to assess the representations is to consider the desiderata enumerated in the introduction. With respect to penetrability, the tactical planner can consult the COA to reason about

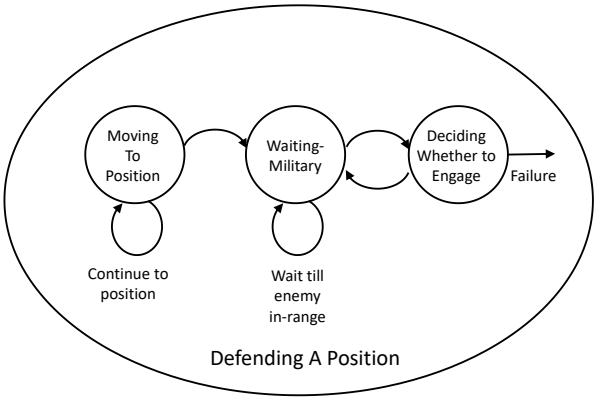

Figure 2. Defending A Position

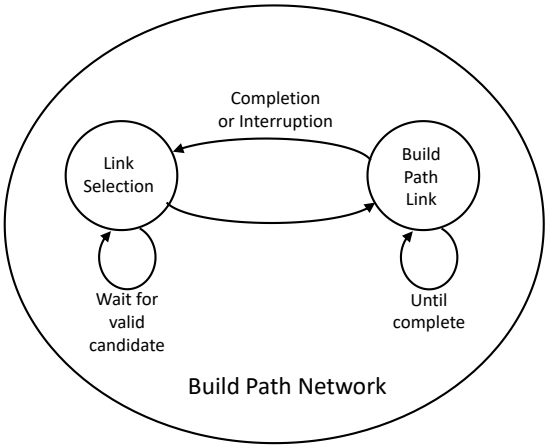

Figure 3. Build Path Network

existing assignments, thereby avoiding overloading a unit with multiple tasks and sidestepping the "sorcerer's apprentice" problem of assigning multiple units to the same task. The representation addresses the criterion for expressiveness, because it supports complex control strategies including branching, looping, and exception handling, rather than simple sequences or macro-operators. We may find that the current technique of applying actions to reschedule and loop would be better handled directly in the agenda, by elaborating sequencing constraints, but we have not yet made that decision. The learnability criterion is addressed by reducing the learning problem to individual decisions, including simple go/no-go decisions, as well as choices among alternatives. This is a much more tractable learning problem than inventing tactics from scratch. Lastly, transferability is addressed through the domain translation mechanisms in the tactical planner. The tactics so far have all been defined in a completely domain-neutral way with a small set of mapping statements to make connections from the game to more general concepts.

6. Related Work

The representations described here result from a long evolution. The instance-level representations are quite similar to what we used in earlier experiments with Tactical Decision Games (TDGs) (Hinrichs et. al, 2005), but those were not operational in that they did not drive actions in a simulation. Their primary function was to support recognition of similar situations and propose courses of action. There, we captured the spatial aspect by a sketched map rather than a grid of tiles.

Our later work on Deep Green/Simpath (Hinrichs et al., 2011) was operational with respect to qualitative envisionment. A key contribution in that work was the qualitative modeling of tactics. While tactic instances were declaratively represented, the process instantiation mechanism was largely hard-coded. In the current work, tactics can be quantity conditioned, but this capability is not yet used. Learning the quantity conditions for effectively employing a tactic is a logical next step.

Other researchers have explored tactics as they apply to non-player characters in game simulations. Van Lent et al. (2004) addressed the problem of explainable AI in terms of recognizing their tactics from execution traces. This was invoked during the after-action review in the training simulation Full Spectrum Command. However, this system had no declarative representation of tactics and the recognition mechanism was specific to the events in the particular simulator.

Ponsen et al. (2006) focused on learning game tactics using a combination of genetic algorithms and reinforcement learning. Their system learned tactical sequences that are specific to the Wargus game, but did not rely on a prior library of general tactics. Our approach explicitly avoids the problem of learning tactical sequences (much less coordinated looping and event-driven behavior), and focuses instead on decomposing tactical behavior to learnable decisions.

Gordon's (2004) library of general strategies is almost the diametric opposite of our game tactic implementations. He collected 372 problem-solving tactics (which he calls strategies) from the perspective of human-like commonsense reasoning. He grouped these into ten domains, such as science, education, animal behavior, business. His military strategies derive from Sun Tzu's Art of War, rather than from modern military doctrine. Given the broad range of tactics, he identified 48 areas or categories of representational elements, such as events, space, and entities. These provided a very broad and abstract framework for representing tactics, but they lacked a concrete task and did not address issues of operationalizing tactics for a simulated agent.

7. Discussion

We have described a representation for tactical knowledge and focused on some aspects that support operationalizing to executable behaviors. Our primary aim was to provide background knowledge to a learning agent in a form that is general, expressive, translates to game-specific actions, and can be reduced to independent, learnable decisions. The declarative, Davidsonian-style representation fits these criteria because each role relation or slot represents a decision for which cases can be gathered, generalized, and consulted during autonomous play. These decision cases may contain domain-specific relations and concepts, rather than the tactical representations themselves. We have, in fact, performed preliminary experiments in learning to defend a position, specifically a go/no-go decision on whether to engage approaching adversarial units. Although the proportion of successful attacks did improve, the results are still too preliminary to report here.

The tactics we have implemented so far have been relatively simple. We have not attempted to replicate the Army field manuals for maneuver warfare or other detailed doctrine; consequently we view them as naïve tactics, by analogy to naïve physics. Future work will likely focus more on other tactical areas in strategy games, such as economics and research, and applying it to other domains than games.

Acknowledgements

We would like to thank the Air Force Office of Scientific Research for supporting this work.

References

Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence*, *24*, 85–168.

Forbus, K., & Hinrichs, T. (2006). Companion cognitive systems: A step towards human-level AI. *AI Magazine* 27(2), 83–95.

Gordon, A. S. (2004). The representation planning strategies. *Artificial Intelligence, 153,* 287–305.

Hinrichs, T. R., & Forbus, K. D. (2016). Qualitative models for strategic planning. *Advances in Cognitive Systems*, 4, 2016, 75–92

Hinrichs, T. R. and Forbus, K. D. (2012). Learning qualitative models by demonstration. *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (pp. 207–213). Toronto, Canada.

Hinrichs, T., Forbus, K., de Kleer, J., Yoon, S., Jones, E., Hyland, R., & Wilson, J. (2011). Hybrid qualitative simulation of military operations. *Proceedings of the Twenty-Third Innovative Applications for Artificial Intelligence Conference* (pp. 1655–1661). San Francisco, CA.

Hinrichs, T., & Forbus, K. (2011). Transfer learning through analogy in games. *AI Magazine* 32(1), Spring 2011, 72–83.

Hinrichs, T. R., & Forbus, K. D. (2007). Analogical learning in a turn-based strategy game. *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (pp. 853–858). Hyderabad, India..

Hinrichs, T. R., Dunham G., & Forbus, K. D. (2005) Analogical learning in tactical decision games. arXiv:2108.08227 [cs.AI].

Mostow, David J. (1881). Mechanical transformation of task heuristics into operational procedures. Doctoral dissertation, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA.

Ponsen, M., Munoz-Avila, H., Spronck, P., & Aha, D. W. (2006). Automatically generating game tactics through evolutionary learning. *AI Magazine*, 27(3), 75–75.

Van Lent, M., Fisher, W., & Mancuso, M. (2004). An explainable artificial intelligence system for small-unit tactical behavior. *Proceedings of the Sixteenth Innovative Applications of Artificial Intelligence* (pp. 900–907). San Jose, CA.