Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment

A Particle Model for State Estimation in Real-Time Strategy Games

Ben G. Weber
Expressive Intelligence Studio
UC Santa Cruz
bweber@soe.ucsc.edu

Michael Mateas
Expressive Intelligence Studio
UC Santa Cruz
michaelm@soe.ucsc.edu

Arnav Jhala
Expressive Intelligence Studio
UC Santa Cruz
jhala@soe.ucsc.edu

Abstract

A big challenge for creating human-level game AI is building agents capable of operating in imperfect information environments. In real-time strategy games the technological progress of an opponent and locations of enemy units are partially observable. To overcome this limitation, we explore a particle-based approach for estimating the location of enemy units that have been encountered. We represent state estimation as an optimization problem, and automatically learn parameters for the particle model by mining a corpus of expert StarCraft replays. The particle model tracks opponent units and provides conditions for activating tactical behaviors in our StarCraft bot. Our results show that incorporating a learned particle model improves the performance of EISBot by 10% over baseline approaches.

Introduction

Video games are an excellent domain for AI research, because they are complex environments with many real-world properties (Laird and van Lent 2001). One of the environment properties under the designer's control is how much information to make available to agents. Designers often limit the amount of information available to agents, because it enables more immersive and human-like behavior (Butler and Demiris 2010). However, accomplishing this goal requires developing techniques for agents to operate in partially observable environments.

StarCraft is a real-time strategy (RTS) game that enforces imperfect information. The "fog of war" limits a player's visibility to portions of the map where units are currently positioned. To acquire additional game state information, players actively scout opponents to uncover technological progress and locations of enemy forces. Players use information gathered during scouting to build expectations of future opponent actions. One of the challenges in RTS games is accurately identifying the location and size of opponent forces, because opponents may have multiple, indistinguishable units.

We investigate the task of maximizing the amount of information available to an agent given game state observations. To accomplish this goal, we propose a particle-based approach for tracking the locations of enemy units that have been scouted. Our approach is inspired by the application of particle filters to state estimation in games (Bererton 2004). It includes a movement model for predicting the trajectories of units and a decay function for gradually reducing the agent's confidence in predictions.

To select parameters for the particle model, we represent state estimation as a function optimization problem. To implement this function, we mined a corpus of expert StarCraft replays to create a library of game states and observations, which are used to evaluate the accuracy of a particle model. Representing state estimation as an optimization problem enabled off-line evaluation of several types of particle models. Our approach uses a variation of the simplex algorithm to find near-optimal parameters for the particle model.

We have integrated the best performing particle models in EISBot, which is a StarCraft bot implemented in the ABL reactive planning language (Weber et al. 2010). The particle model provides conditions that are used to activate tactical behaviors in EISBot, such as defending a base or engaging enemy forces. We evaluate the performance of the particle-based approach against the built-in AI of StarCraft, as well as entries from the AIIDE 2010 StarCraft AI competition. The results show that the optimized particle model improved both the win ratio and score ratio of EISBot by over 10% versus baseline approaches.

Related Work

There are two main approaches for estimating the position of a target which is not visible to an agent. In a space-based model, the map is represented as a graph and each vertex is assigned a probability that it contains the target. In a particle-based model, a cloud of particles represent a sampling of potential coordinates of the target (Darken and Anderegg 2008). Both approaches can apply a movement model for updating estimations of target locations.

Occupancy maps are a space-based model for tracking targets in a partially observable environment (Isla 2006). The map is broken up into a grid, where each node is the grid is connected to adjacent nodes. During each update there is a diffusion step where each node transfers a portion of the probability it contains the target uniformly to adjacent nodes. The update cycle contains a visibility check where nodes visible to the agent not containing the target are assigned a weight of zero, and a normalization process that scales the weights of nodes. One of the challenges in applying occupancy maps is selecting a suitable grid resolution, because visibility computations can become prohibitively expensive on large grids.

Tozour (2004) presents a variation of occupancy maps which incorporate observations made by an agent. The agent maintains a list of visited nodes, and searches for targets by exploring nodes that have not been investigated. The model can incorporate a decay, which will cause the agent to gradually investigate previously explored nodes.

Hladky and Bulitko (2008) demonstrate how the accuracy of occupancy maps can be improved by applying movement models based on player behavior. Rather than uniformly transfer probability to adjacent nodes, their approach uses hidden semi-Markov models (HSMM) to learn transitions between grid nodes based on previous observations. The transitions are learned by analyzing a corpus of game logs, extracting game state and observations, and training motion models from this data. While the accuracy of predictions generated by the HSMMs were comparable with human experts, their analysis was limited to a single map.

Particle filters are an alternative method for tracking targets. This approach has a rich history in robotics, and has been applied to several problems including localization and entity tracking (Thrun 2002). The application of particle filters to state estimation in games was first proposed by Bererton (2004) as a technique for creating agents that exhibit an illusion of intelligence when tracking targets. Particle filters can be applied to target tracking by placing a cloud of particles at the target's last known position, where each particle performs a random walk and represents a potential target location. Each update cycle, the position of each particle is updated based on a movement model, particles visible by the agent are removed from the list of candidate target locations, and the weights of the particles are normalized.

One way of improving the accuracy of particle filters is to apply movement models that mimic the behavior of the target they represent. Darken and Anderegg (2008) refer to particles that imitate agent or player behavior as simulacra, and claim that simulacra result in more realistic target tracking. They propose candidate movement models based on different types of players. Another approach for improving accuracy is to replace the random walk behavior with complex movement models that estimate the paths of targets (Southey, Loh, and Wilkinson 2007).

Particle Model

The goal of our model is to accurately track the positions of enemy units that have been previously observed. Our approach for achieving this task is based on a simplified model of particle filters. We selected a particle-based approach instead of a space-based approach based on several properties of RTS games. It is difficult to select a suitable grid resolution for estimation, because a tile-based grid may be too fine, while higher-level abstractions may be too coarse. Also, the model needs to be able to scale to hundreds of units. Finally, the particle model should be generalizable to new maps.

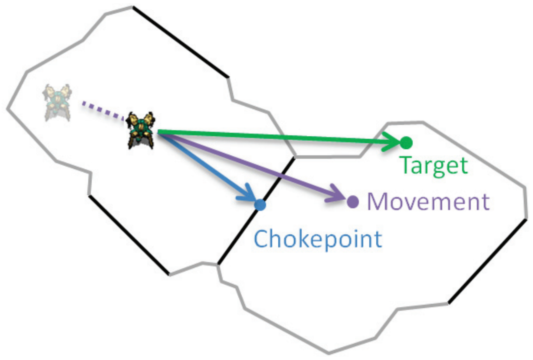

Figure 1: Particle trajectories are computed using a linear combination of vectors. The movement vector captures the current trajectory, while the target vector factors in the unit's destination and the chokepoint vector factors in terrain.

Our particle model is a simplified version of particle filters, where a single particle is used to track the position of a previously encountered enemy unit, instead of a cloud of particles. A single particle per unit approach was chosen, because an opponent may have multiple, indistinguishable units. Since an agent is unable to identify individuals across multiple observations, the process for culling candidate target locations becomes non-trivial. We address this problem by adding a decay function to particles, which gradually reduces the agent's confidence in estimations over time.

Particle Representation

Particles in our model are assigned a class, weight, and trajectory. The class corresponds to the unit type of the enemy unit. Our system includes the following classes of units: building, worker unit, ground attacker, and air attacker. Each class has a unique set of parameters used to compute the trajectories and weights of particles.

Particles are assigned a weight that represents the agent's confidence in the prediction. A linear decay function is applied to particles in which a particle's weight is decreased by the decay amount each update. Particles with a weight equal to or less than zero are removed from the list of candidate target locations. Different decay rates are used for different classes of particles, because predictions for units with low mobility are likely to remain accurate, while predictions for units with high mobility can quickly become inaccurate.

Each particle is assigned a constant trajectory, which is computed based on a linear combination of vectors. A visualization of the different vectors is shown in Figure 1. The movement vector is based on observed unit movement, which is computed as the difference between the current coordinates and previous coordinates. Our model also incorporates a chokepoint vector, which enables terrain features to be incorporated in the trajectory. It is found by computing the vectors between the unit's coordinates and the center point of each chokepoint in the current region, and selecting the vector with the smallest angle with respect to the movement vector. The target vector is based on the unit's destination and is computed as the difference between the destination coordinates and current unit coordinates. Computing the target vector requires accessing game state information that is not available to human players.

The trajectory a of particle is computed by normalizing the vectors to unit vectors, multiplying the vectors by class-specific weights, and summing the resulting vectors. Our model incorporates unique movement and target weights for each particle class, while a single weight is used for the chokepoint vector.

Update Process

The particle model begins with an initially empty set of candidate target locations. As new units are encountered during the course of a game, new particles are spawned to track enemy units. The model update process consists of four sequential steps:

- **Apply movement:** updates the location of each particle by applying the particle's trajectory to its current location.
- **Apply decay:** linearly decreases the weight of each particle based on its class.
- **Cull particles:** removes particles that are within the agent's vision range or have a less than zero weight.
- **Spawn new particles:** creates new particles for units that were previously within the agent's vision range that are no longer within the agent's vision range.

The spawning process instantiates a new particle by computing a trajectory, assigning an initial weight of one, and placing the particle at the enemy unit's last known position.

Unlike previous work, our model does not perform a normalization process, because multiple units may be indistinguishable. Additionally, our model does not commit to a specific sampling policy. The process for determining which particles to sample is left up to higher-level agent behaviors.

Model Training

We first explored the application of particle models to StarCraft by performing off-line analysis. The goal of this work was to determine the accuracy of different model settings and to find optimal trajectory and decay parameters for the models. To evaluate the models, we collected a corpus of StarCraft replays, extracted game state and observation data from the replays, and simulated the ability of the models to predict the enemy threat in each region of the map at each timestep.

Data Collection

To enable off-line analysis of particle models, we collected thousands of expert-level StarCraft replays from tournaments hosted on the International Cyber Cup[1]. We sampled the replays by randomly selecting ten replays for each unique race match up. An additional constraint applied during the sampling process was that all replays in a sample were played on distinct maps. This constraint was included to ensure that the particle models are applicable to a wide variety of maps.

We extracted game state information from the sampled replays by viewing them using the replay mode of StarCraft and querying game state with the Brood War API[2]. Our replay tool outputs a dump of the game state once every 5 seconds (120 frames), which contains the positions of all units. The extracted data provides sufficient information for determining which enemy units are visible by the player at each timestep. The resulting data set contains an average of 2,852 examples for each race match up.

Error Function

We present a region-based metric for state estimation in StarCraft, where the role of the particle model is to predict the enemy threat in each region of the map. Our error function makes use of the Brood War Terrain Analyzer, which identifies regions in a StarCraft map (Perkins 2010). The particle model is limited to observations made by the player, while the error function is based on complete game state.

Error in state estimation can be quantified as the difference between predicted and actual enemy threat. Our particle model predicts the enemy threat in each region based on the current game state and past observations. For each region, the enemy threat is computed as the number of visible enemy units in the region (unit types are uniformly weighted), plus the summation of the weights of particles within the region. Given predictions of enemy threat at timestep t, we compute state estimation error as follows:

$$error(t) = \sum_{r \in R} |p(r,t) - a(r,t)|$$

where $p(r,t)$ is the predicted enemy threat of region r at timestep t, $a(r,t)$ is the actual number of enemy units present in region r at timestep t, and R is the set of regions in a map. The actual threat for a region can be computed using the complete information available in the extracted replay data. The overall error for a replay is defined as follows:

$$error = \frac{1}{T} \sum_{t=1}^{T} error(t)$$

where T is the number of timesteps, and *error* is the average state estimation error.

Parameter Selection

Our proposed particle model includes several free parameters for specifying the trajectories and decay rates of particles. To select optimal parameters for the particle model, we represent state estimation as an optimization problem: the state estimation error serves as an objective function, while the input parameters provide a many-dimensional space. The set of parameters that minimizes our error function is selected as optimal parameters for our particle model.

To find a solution to the optimization problem, we applied the Nelder-Mead technique (Nelder and Mead 1965), which is a downhill simplex method. We used Michael Flanagan's minimization library[3] which provides a Java implementation

[1]http://iccup.net

[2]http://code.google.com/p/bwapi

[3]http://www.ee.ucl.ac.uk/~mflanaga/java

Table 1: The accuracies of the different particle models varies based on the specific race match up. Overall, the optimized particle model performed best in the off-line state estimation task. Providing the particle models with additional features, including the target vector (T) and ability to distinguish units (I), did not improve the overall accuracies.

	PvP	PvT	PvZ	TvP	TvT	TvZ	ZvP	ZvT	ZvZ	Overall
Default	0.751	0.737	0.687	0.571	0.781	0.826	0.628	0.684	0.357	0.669
Default$_I$	0.749	0.728	0.710	**0.721**	0.762	0.766	0.682	0.697	0.561	0.709
Optimized	**0.841**	**0.772**	**0.733**	0.709	**0.810**	**0.827**	**0.697**	**0.722**	0.544	**0.739**
Optimized$_I$	0.749	0.728	0.710	**0.721**	0.762	0.767	0.682	0.698	**0.563**	0.709
Optimized$_T$	**0.841**	0.740	0.712	0.709	**0.810**	**0.827**	**0.697**	**0.722**	0.544	0.733

of this algorithm. The stopping criterion for our parameter selection process was 500 iterations, providing sufficient time for the algorithm to converge.

Evaluation

We compared the performance of our particle model with a baseline approach as well as a perfect prediction model. The range of values between the baseline and theoretical models provides a metric for assessing the accuracy of our approach. We evaluated the following models:

- **Null Model:** A particle model that never spawns particles, providing a baseline for worst-case performance.
- **Perfect Tracker:** A theoretical model which perfectly tracks units that have been previously observed, representing best-case performance.
- **Default Model:** A model in which particles do not move and do not decay, providing a last known position.
- **Optimized Model:** Our particle model with weights selected from the optimization process.

The null model and perfect tracker provide bounds for computing the accuracy of a model. Specifically, we define the accuracy of a particle model as follows:

$$accuracy = \frac{error_{NullModel} - error}{error_{NullModel} - error_{PerfectTracker}}$$

where *error* is the state estimation error. Accuracy provides a metric for evaluating the ability of a particle model to estimate enemy threat.

The accuracy of the default and optimized models for each of the race match ups are shown in Table 1. A race match up is a unique pairing of StarCraft races, such as Protoss versus Terran (PvT) or Terran versus Zerg (TvZ). The table also includes results for variations of the particle models which were provided with additional features. The Default$_I$ and Optimized$_I$ models were capable of identifying specific enemy units across observations, and the Optimized$_T$ model used the target vector while other models did not. Accuracies for the null model and perfect tracker are not included, because these values are always 0 and 1. Overall, the optimized particle model, which is limited to features available to humans, performed best. Providing additional information to the particle models did not, on average, improve the accuracy of models.

We also investigated the variation in accuracy of different models over the duration of a game, which provides some insights into the scouting behavior of players. The average threat prediction errors for the different models in Terran versus Protoss games is shown in Figure 2. In this race match up, there was a noticeable difference between the accuracies of the default and optimized models. Players tend to scout the opponent between three and four minutes game time, which leads to improved state estimations. There is little difference between the default and optimized particle models in the first 12 minutes of the game, but the optimized model is noticeably more accurate after this period.

Table 2: Decay rates for the different particle classes in Protoss versus Zerg games.

	Decay Rate	Lifetime (s)
Building	0.00	∞
Worker	0.00	∞
Ground Attacker	0.04	22.22
Air Attacker	0.13	5.78

Table 3: Weights for the movement and chokepoint vectors in Protoss versus Zerg games, in pixels per second.

	Movement Vector
Building	0.00
Worker	5.67
Ground Attacker	5.35
Air Attacker	31.57
	Chokepoint Vector
All Classes	20.96

The parameter sets that resulted in the highest accuracy for state prediction in Protoss versus Zerg games, which are used by our agent, are shown in Table 2 and Table 3. As expected, buildings have a decay rate of zero, because the majority of buildings in StarCraft are immobile. Units that tend to remain in the same region, such as worker units, have a long lifetime, while highly mobile units that can quickly move between regions have short lifetimes. The lack of building movement is also indicated by the movement vector. For ground attacking units, the chokepoint vector was the highest weighted vector, while for air attacking units, the movement vector was the highest weighted vector.

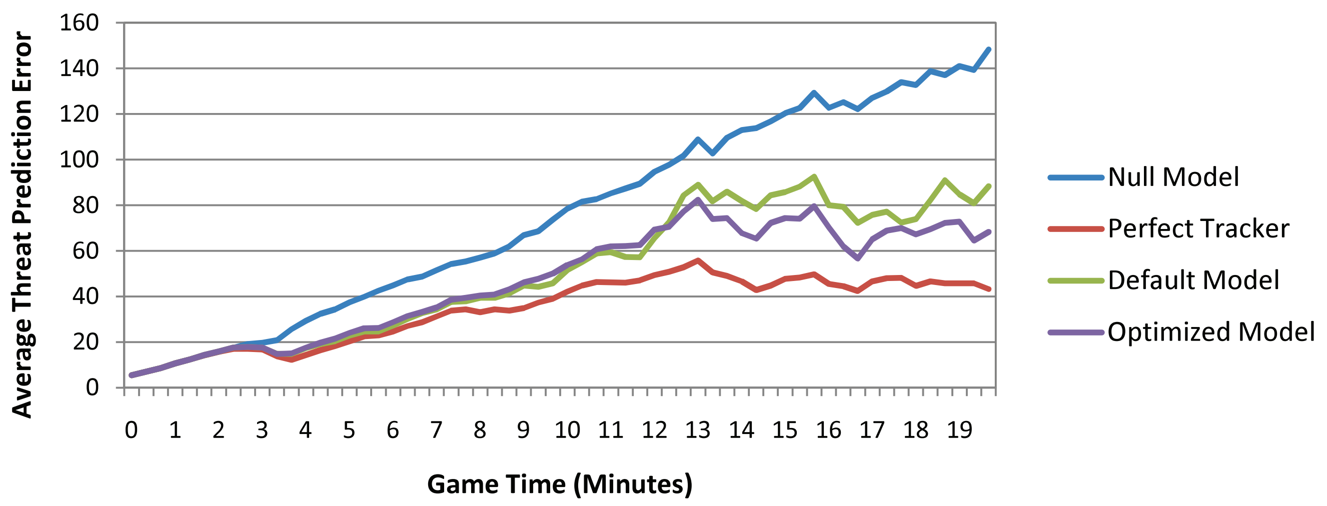

Figure 2: The average error of the particle models in Terran versus Protoss games vary drastically over the duration of a game. The accuracy of the particle models improve over baseline performance once enemy units are scouted. The optimized particle model noticeably outperforms the default particle model after 12 minutes.

Implementation

We selected the best performing models from off-line analysis of state estimation and integrated them into EISBot, which is our Protoss bot for the AIIDE 2011 StarCraft AI Competition[4]. EISBot is a reactive planning agent composed of several managers that specialize in specific aspects of gameplay. The tactics manager is responsible for deciding when and where to attack opponent forces. It uses state estimations from the particle model in the following tactics behaviors:

- **Defend base:** assigns idle attacker units to defensive locations based on particle positions.
- **Attack target:** selects locations to attack based on particle positions.

EISBot executes the particle model update process each game cycle. New particles that have been spawned are placed into working memory, enabling reactive planning behaviors to use predictions in behavior condition checks. To populate EISBot's working memory, scouting behaviors have been added to the agent to ensure that it encounters enemy units. An overview of the additional competencies in EISBot is available in previous work (Weber et al. 2010).

We investigated four particle model settings in EISBot. These mirror the models used in off-line analysis with one modification: the perfect tracker was replaced by a perfect information model, which is granted complete game state. With the exception of the particle model policy, all other EISBot settings and behaviors were held fixed.

EISBot was evaluated against the built-in AI of StarCraft as well as bots from the 2010 AI competition. We selected bots that incorporate flying units into their strategies, which includes Skynet (Protoss), Chronos (Terran), and Overmind (Zerg). EISBot faced a total of six opponents: each race of the built-in AI and the three competition bots. The map pool consisted of a subset of the maps used in the competition: Python and Tau Cross.

Each model was evaluated against all opponent and map permutations in three game match ups, resulting in a total of 144 games. Win rates for the different models are shown in Table 4 and score ratios are shown in Table 5. Score ratio is defined as EISBot's score divided by the opponent's score averaged over the set of games played and provides a finer resolution of performance than win rate. Overall, the optimized particle model had both the highest win rates and score ratios by over 10%. The optimized particle model had the same win rates on both of the maps, while the default model performed 10% better on Tau Cross and the perfect information model performed 10% better on Python.

[4]http://StarCraftAICompetition.com

Table 4: Win rates against the other bots with different particle model settings.

	Protoss	Terran	Zerg	Overall
Perfect Info.	50%	**83%**	67%	67%
Null Model	50%	75%	75%	67%
Default Model	58%	75%	67%	67%
Optimized Model	**75%**	75%	**83%**	**78%**

Table 5: Score ratios against the other bots with different particle model settings.

	Protoss	Terran	Zerg	Overall
Perfect Info.	1.26	1.54	1.35	1.38
Null Model	1.06	**1.58**	1.59	1.41
Default Model	1.16	1.46	1.35	1.32
Optimized Model	**1.55**	1.52	**1.60**	**1.56**

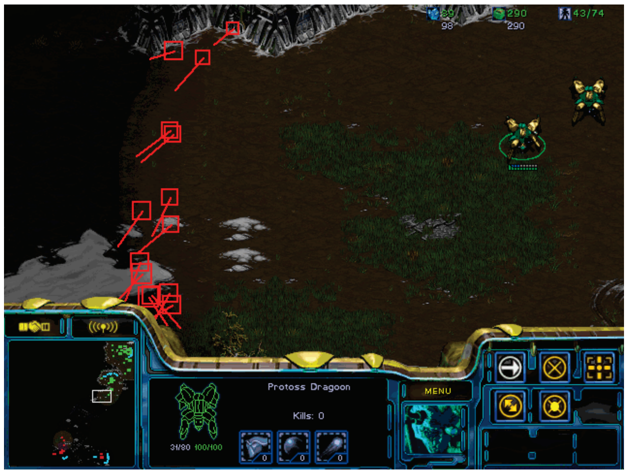

Figure 3: A screen capture of an EISBot versus Overmind game showing predicted locations and trajectories of flying units no longer in EISBot's vision range.

The optimized model had the highest win rate against Overmind, the previous competition winner. It won 67% of matches against Overmind while the other models had an average win rate of 50% against Overmind. A screen capture visualizing the optimized particle model tracking Overmind's mutalisks is shown in Figure 3. The particle trajectories are used to anticipate the future positions of the enemy mutalisks.

A surprising result was that the perfect information model did not perform best, since it has the most accurate information about the positions of enemy forces. The most likely cause of this result was the lack of scouting behavior performed when utilizing this model. Since the agent has perfect information, it does not need to scout in order to populate working memory with particles. Scouting units improved the win rate of the agent by distracting the opponent, such as diverting rush attacks.

Conclusions and Future Work

We have introduced a model for performing state estimation in real-time strategy games. Our approach is a simplified version of particle filters that incorporates a constant trajectory and linear decay function for particles. To evaluate the performance of different particle models, we extracted game state and observation data from StarCraft replays, and defined metrics for measuring accuracy. These metrics were also applied to an optimization problem, and used to find optimal parameter settings for the particle model.

The particle model was integrated in EISBot and evaluated against a variety of opponents. Overall, the optimized particle model approach outperformed the other models by over 10%. Our results also showed that making more game state available to the agent does not always improve performance, as the ability to identify specific units between observations did not improve the accuracy of threat prediction.

While the optimized model outperformed the default model, there is still a large gap between it and the perfect tracker. Future work could explore more complex movement models for particles, including simulacra (Darken and Anderegg 2008), path prediction (Southey, Loh, and Wilkinson 2007), or models that incorporate qualitative spatial reasoning (Forbus, Mahoney, and Dill 2002). Additionally, future work could investigate particle models that more closely resemble particle filters.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Number IIS-1018954. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Bererton, C. 2004. State Estimation for Game AI Using Particle Filters. In *Proceedings of the AAAI Workshop on Challenges in Game AI*. AAAI Press.

Butler, S., and Demiris, Y. 2010. Partial Observability During Predictions of the Opponent's Movements in an RTS Game. In *Proceedings of CIG*, 46–53. IEEE Press.

Darken, C., and Anderegg, B. 2008. *Game AI Programming Wisdom 4*. Charles River Media. chapter Particle Filters and Simulacra for More Realistic Opponent Tracking, 419–427.

Forbus, K. D.; Mahoney, J. V.; and Dill, K. 2002. How Qualitative Spatial Reasoning can Improve Strategy Game AIs. *IEEE Intelligent Systems* 25–30.

Hladky, S., and Bulitko, V. 2008. An Evaluation of Models for Predicting Opponent Positions in First-Person Shooter Video Games. In *Proceedings of CIG*, 39–46. IEEE Press.

Isla, D. 2006. *Game AI Programming Wisdom 3*. Charles River Media. chapter Probabilistic Target Tracking and Search Using Occupancy Maps, 379–388.

Laird, J. E., and van Lent, M. 2001. Human-level AI's Killer Application: Interactive Computer Games. *AI magazine* 22(2):15–25.

Nelder, J. A., and Mead, R. 1965. A Simplex Method for Function Minimization. *The Computer Journal* 7(4):308–313.

Perkins, L. 2010. Terrain Analysis in Real-time Strategy Games: Choke Point Detection and Region Decomposition. In *Proceedings of AIIDE*, 168–173. AAAI Press.

Southey, F.; Loh, W.; and Wilkinson, D. 2007. Inferring Complex Agent Motions from Partial Trajectory Observations. *Proceedings of IJCAI*.

Thrun, S. 2002. Particle Filters in Robotics. In *Proceedings of Uncertainty in AI*.

Tozour, P. 2004. *Game AI Programming Wisdom 2*. Charles River Media. chapter Using a Spatial Database for Runtime Spatial Analysis, 381–390.

Weber, B. G.; Mawhorter, P.; Mateas, M.; and Jhala, A. 2010. Reactive Planning Idioms for Multi-Scale Game AI. In *Proceedings of CIG*, 115–122. IEEE Press.