Simulating Aggregate Player Behavior With Learning Behavior Trees

Emmett Tomai
Rosendo Salazar
Roberto Flores
University of Texas – Pan American
1201 W. University Ave.
Edinburg, TX 78539
tomaie@utpa.edu, rsalazar6@utpa.edu, rfloresx@utpa.edu

Keywords:
Virtual worlds, machine learning, player modeling, behavior trees

ABSTRACT: *In this paper we investigate the use of Behavior Trees and machine learning to train behavioral controllers for human-like agents in a Massively Multiplayer Online Role-Playing Game. A Learning Behavior Tree, based on the AI scripting technique from the video game industry, is used to model the overall behavior of the agents and to control training and execution of a learned sub-behavior (setting waypoints for more human-like movement between regions of interest). We provide empirical evidence that agents using the learned controller show similarity in aggregate to human player populations in the same environment. We argue that Learning Behavior Trees are a useful, general purpose approach for learning sub-behaviors within a larger behavior.*

1. Introduction

Online virtual worlds are an increasingly significant venue for human interaction. By far the most active virtual worlds are video games, headlined by the explosive success of Blizzard Entertainment's Massively Multiplayer Online Role-Playing Game (MMORPG) *World of Warcraft (WoW)* and hugely ambitious follow ups like BioWare's *Star Wars: The Old Republic*. With growing awareness of the potential of these embodied, online interactions, there is increasing interest in virtual worlds for education, training and scientific research as well (cf. Bainbridge, 2006; Dickey, 2005).

Due to the shared, persistent nature of these virtual worlds, user behaviors and experiences are shaped by aggregate population dynamics. An MMORPG designer must be conscious of how each design decision will impact things like where players congregate in the world, the demand on shared resources and how they will help or impede one another. These issues often cause unforeseen problems in games, and could easily derail research or training projects that cannot afford to simply try again later. In this work we are investigating agent control models to mimic the aggregate behavior of MMORPG players. These agents could ultimately be used to explore design decisions, automate pre-testing before expensive live tests, or to make the world feel more full and alive for human players. In this work we focus on the movement of agents as they pursue in-game goals. We introduce the *Learning Behavior Tree* control model and propose that it is a useful, general purpose approach to learning behaviors in the context of more complex behaviors.

2. Related Work

Simulating human-like crowds of pedestrians has been explored by Shao using a full-agent approach combining reactive controllers, scripted behaviors and mental states with path-planning and goals (Shao & Terzopoulos, 2005). In contrast, Treuille proposed a deliberately simpler model with less individual variation but less cost in modeling and run-time resource requirements (Treuille et. al, 2006). In both cases the goal was human-like crowd simulation based on local collision avoidance and individual global navigation. Players in a virtual world do not move around like normal pedestrians, so the

emphasized features such as local collision avoidance are not strongly applicable here. But that work suggests that human-like simulation should pursue integration of reactive, scripting and planning behaviors.

Automated learning techniques have been applied to learn human-like movement for virtual world agents. Henry trained an agent controller using *Inverse Reinforcement Learning* to navigate through crowded environments, making the distinction between shortest and human-like paths (Henry, 2010). Togelius evaluated several co-evolution strategies for creating car racing controllers with the aim of deploying a diverse population of human-like AI opponents in a car racing game (Togelius et. al, 2007). In the *first-person shooter (FPS)* video game domain, Geisler notes the high predictability and manual labor involved in traditional AI scripting of game agent opponents (*bots*) as motivation for automatic learning of human-like behavior (Geisler, 2004). Several approaches have been applied to learn human-like movement and facing. Thurau used *Self-Organizing Maps* and *Artificial Neural Networks* to learn based on position and relative enemy positions (Thurau et. al, 2003) while Geisler evaluated *Naïve Bayes* and neural network approaches (Geisler, 2004) with promising results. These and numerous other results (cf. Galway et. al, 2008) have demonstrated that machine learning and evolutionary computation are well suited to optimizing problems that have a reactive nature (e.g. positioning relative to other agents, strategic responses), a small number of output dimensions (e.g. movement and facing) and work at a single level of abstraction (Bakkes et. al, 2012). However, moving to more complex behaviors requires working at multiple levels of abstraction (Bakkes et. al, 2012). Several established cognitive architectures have been applied to the problem of learning goal-based movement in games. Soar was proposed for creating synthetic adversaries in the MOUT (Military Operations on Urbanized Terrain) domain, emphasizing believability and diversity as we do here (Wray et. al, 2005). It was evaluated on its ability to show transfer learning for different goal locations and topologies (Gorski & Laird, 2006). Best detailed how ACT-R could be used in the same domain with lower-level perceptual input only (Best et. al, 2002). Both systems learn from experience how to accomplish a certain goal. By contrast, we are beginning with a human-authored script and learning from examples how to refine it to better fit human-like behavior. Several approaches have found success by combining learned behaviors with human-encoded knowledge. Spronck has applied *Dynamic Scripting* to both group combat in the *Role-Playing Game (RPG)* genre and strategic decision-making in the *Real-Time Strategy (RTS)* genre (Spronck et.al, 2006). A knowledge base of manually created rules is combined with learning inclusion and ordering of those rules into scripts. Marthi used *Hierarchical Reinforcement Learning* for learning joint movement of units in the RTS domain (Marthi et. al, 2005). The reinforcement learning of movement is embedded in a manually created *concurrent ALisp* program. The program encodes knowledge about the task context and controls both the training and execution of the learned behaviors in that context. We propose a similar approach in this work, with a more explicit, declarative composition. Finally, Schrum has created a FPS bot architecture that learns combat behavior using *Neuroevolution* (Schrum et. al, 2012) and won the 2K Games' 2012 BotPrize while being judged as human more than 50% of the time (Karpov et. al, 2012). The learned combat behavior is one component of the architecture, organized in a *Behavior Tree*-like structure that encodes human intuition about the priority and trigger conditions for that behavior and others. In this work we look more generally at Behavior Trees as a flexible controlling architecture for mixing learned and procedural behaviors.

3. MMORPG Player Behavior

The evaluation domain for this work is player movement between regions of interest in a MMORPG. In MMORPGs, players control avatar characters in a physically simulated virtual world that is both shared and persistent. In contrast to more reactive and/or linear environments in other genres, players roam freely in the world, picking up tasks and completing them at their own discretion. There are different regions where each task is acquired, completed and turned in for credit. Thus, players have goal-directed travel between those regions of interest. The player freedom to pursue any number of tasks or none at all in the shared space makes the

population dynamics difficult to predict, as each players' actions impacts the experience of all the players around them in a feedback loop. This work seeks to generate populations of independent agents in an MMORPG world that exhibit human-like aggregate behavior, specifically in how the population moves between points of interest. Other behaviors such as task decision making are not addressed in the scope of these experiments.

To collect player behavior data, we created a light weight, research focused MMORPG. We gathered data from 37 human players playing together in a laboratory setting. The game collects a wide range of data for each player, including movement, avatar actions, UI actions and visibility of other entities. The experimental map was divided into separate areas with similar but different topology and tasks to perform. 20 of those players successfully completed the A area of the map (data set A) while 17 others successfully completed the B area (data set B).

4. Learning Behavior Trees

Behavior Trees are a technique for controlling video game AI agents that has gained considerable popularity, notably after use in Bungie's *Halo* series (Isla, 2005). Reusable, procedural behaviors are composed into trees using non-leaf composition nodes that explicitly specify traversal semantics. The key advantage of Behavior Trees, from a game AI point of view, is that non-programmers can create new behaviors through graphical composition, and those trees can be reused as sub-behaviors in other trees. More deeply, this is powerful because using only a small set of well-defined composition nodes, Behavior Trees can integrate different behavioral architectures from purely parallel reactive behaviors to purely sequential decision trees. While this can be and is done using arbitrary finite state machines in game AI, using principled, declarative composition means that the behaviors themselves do not encode external transitions to other behaviors (Champandard, 2008). We propose that Behavior Trees offer a similar advantage to learning. Within a Behavior Tree, manually written to match a complex behavior, various component behaviors may be learned at various levels of abstraction. The structure of the Behavior Tree encodes not just how those components are used within the more complex overall behavior, but also how they can be trained in the same environment.

When a Behavior Tree is instantiated as the controller for a particular agent, its leaf node behaviors are bound to that agent, and the tree is recursively updated with each simulation step. Each sub-tree that is updated returns a *Success*, *Failure* or *Running* condition that impacts the traversal of the parent node. In many Behavior Tree systems, the tree nodes share data on a common blackboard contained by the agent. For example, the Behavior Tree in Figure 4.1 specifies a traveling behavior to a given *target travel location*.

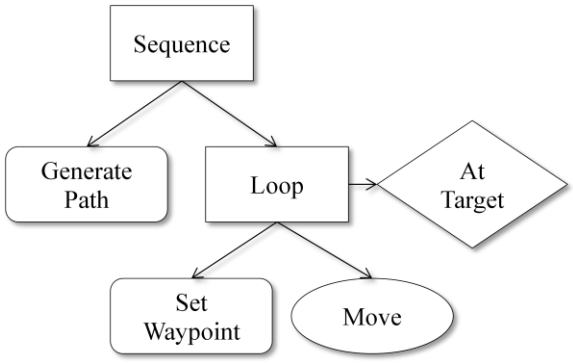

Figure 4.1: Behavior Tree for traveling to a given target location.

The root node is a *Sequence*. Sequences update their children in order until one returns Failure or they all return Success. If a child returns Running, the sequence returns Running, and on the subsequent update, traversal resumes with that child. The Generate Path behavior in the example procedurally generates a path from the agent's location to the travel target location (stored on the blackboard) or returns Failure if no such location exists or it cannot be reached. The next child of the Sequence is a sub-tree which is a path following behavior. Its root node is a *Loop*. Loops update each child in order until one returns Failure, looping back to the beginning each time the last child returns Success. This type of Loop has a special conditional child, shown as a diamond, which is updated at the beginning of each loop. If that child returns Success, the Loop returns Success, otherwise it continues on. The Set Waypoint behavior in the example expects an active path on the

blackboard, and procedurally sets the next point in that path as the *immediate movement target location* (distinct from the travel target location). The Move behavior procedurally moves the agent incrementally closer to that immediate target with each location, until it arrives and returns Success. The At Target behavior is an environmental condition that checks to see if the agent is at the travel target location. If so, it returns Success, the Loop returns Success and the Sequence returns Success. Otherwise, the Loop continues to set and achieve waypoints along the way. This example shows how the composition structure controls the reusable leaf behaviors, including choices like how often the path should be regenerated, how waypoints are handled and when the target location should be checked.

However, the problems of brittleness and predictability are the same as for other hand-authored scripts. To increase variation, we use machine learning to refine individual behaviors within the context of the overall behavior. We implement this with a *Decorator* node. Decorators have a single child and are used to control that child or manipulate its return value. In Figure 4.2 we have placed a Learn on Success Decorator over the Move behavior.

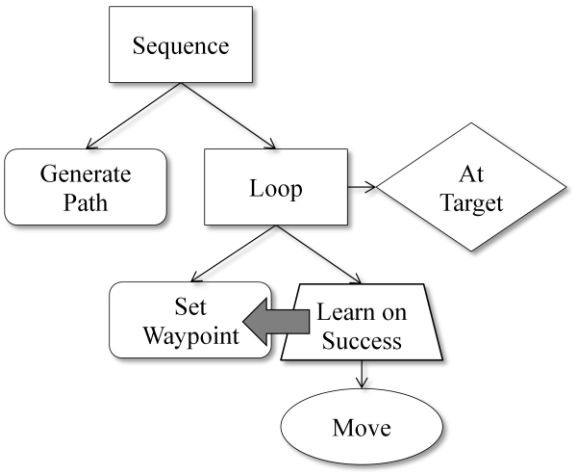

Figure 4.2: Learning Behavior Tree for traveling to a given target location and learning to set waypoints.

Whenever the Move returns Success, indicating it has reached a waypoint, this Decorator invokes a learning routine on a modified Set Waypoint behavior. To train the behavior, an agent is instantiated with this Learning Behavior Tree and the trace of movement data from a single player. In a training episode, a target travel location is taken from the trace, and the learning agent attempts to travel there by updating the Behavior Tree. The learning method is defined by a tuple *(L,E,F,M)* where *L* is the set of possible locations, *E* is the error function between the current waypoint location set by the behavior *l1* and the current location actually traveled to in the training data *l2* (where $l1, l2 \in L$), *F* is a function that returns a set of environmental features and *M* is a learning model that is trained on *E*, *F* to estimate *l2* given *l1*, *F*. The placement of the Decorator within the structure of the tree specifies the frequency and conditions under which the learning should be invoked. Once trained, the Set Waypoint behavior perturbs each waypoint stochastically according to the probabilities learned by *M*, to attempt to create more human-like behavior.

5. Experimental Setup

The goal of these experiments is to evaluate the effectiveness of the Learning Behavior Tree on the task of simulating a human-like population of players in a virtual world. Specifically, modeling travel between points of interest with aggregate similarity to player data. Player locations have been abstracted to a 2-d grid overlaying the world map, as is common in path-finding algorithms.

Each experimental condition involves a certain set of player movement traces (the data set) and a certain agent control model (the model). The data set is divided into equal training and testing sets. For training, an agent using the model is created for each trace in the training data set. Those agents are all simulated to the length of the longest trace in the set, resulting in a single trained model of the entire population (not individual behavior). Then, a set of agents using the trained model is created. This set has the same number of agents as traces in the testing data set, but does not use the data itself. The agents are simulated, and the resulting population dynamics are compared against a simulation of agents replaying the testing player traces (the Reference simulation).

Three agent control models are investigated. The Learning Behavior Tree agent control model works as described in the prior section. In these

experiments, it uses a Naïve Bayes classifier for *M*, appropriate for the small size of the data set. The set of player locations *L* has been abstracted to grid membership. The error function *E* relies on the path generated by the Generate Path behavior to transform grid locations taken from the training data into path-relative steps: forward along the path, right, left or back. The feature extractor *F* generates Boolean features for those relative positions indicating whether they have already been visited or not during the current travel segment. The second agent control model used is the Learning Position/Goal model. It uses the same Naïve Bayes classifier, but without the structure and functionality of the Behavior Tree. It simply trains on the current position, the goal position and the visited state of the adjacent cells, labeled with the absolute location of the next cell moved to in the training trace. In testing, the next step is chosen stochastically based on the learned probabilities. The third agent control model used is the Training Set model, which simply replays a trace from the training set. This model is used to establish a baseline of how similar the training and testing populations are, according to the metrics used.

To measure the similarity between two agent simulations, we compute the distributional similarity, the difference in path length and the difference in percent revisits. Distributional similarity is a measure of how similarly the two populations are distributed throughout the world. From a player point of view, this is whether the crowds are where they are expected to be. For a single step in the simulation, the error for population distribution is calculated as the number of agents in the Reference simulation that are not matched (by location) by an agent in the simulation being tested. The test and Reference simulation always have the same number of agents. For the set of Reference agents A^r and the population distribution function *D* that returns the number of agents in the specified grid cell *i*, the single step error formula is given in (1). For entire simulation runs, the mean percentage error over all steps is reported.

$$e = |A^r| - \sum_{i=1}^{n} min\ (\ D^r(i), D(i)\ ) \qquad (1)$$

Path length and revisits are taken as local measures of how similar the travel segments in the test simulation are to those in the Reference simulation. Because individual agents in the former are not mapped to individual agents in the latter, the total path length and the total percent of cells revisited (within a travel segment) for an agent are averaged over all the agents in a simulation. The difference for that run is simply the difference between test and Reference. For the set of Reference agents A^r and the set of test agents *A*, and the function *p* that returns the total path length for an agent, the formula for the average path length difference for a simulation run is given in (2).

$$e = \frac{1}{n}\sum_{i=1}^{n} p(A_i^r) - \frac{1}{m}\sum_{i=1}^{m} p(A_i) \qquad (2)$$

For the set of Reference agents A^r and the set of test agents *A*, and the function *r* that returns the total percent of cells revisited for an agent, the formula for the percent revisited difference for a simulation run is given in (3).

$$e = \frac{1}{n}\sum_{i=1}^{n} r(A_i^r) - \frac{1}{m}\sum_{i=1}^{m} r(A_i) \qquad (3)$$

In experiment 1, the training and testing sets are randomly, equally divided out from a single data set. The results from the Training Set model provide a baseline of normal variation between human players performing the same tasks in the same area. We predict that the Learning Behavior Tree and Learning Position/Goal models will perform at least as well according to our similarity measures. To provide evidence of generality, three data sets are used in all experiments: the A set, the B set and the two combined (All).

In experiment 2, the sample data is divided in half longitudinally, with the earlier half for training and the later half for testing. This tests the models on the same population in the same area performing different (later) tasks. The results from the Training Set model for distributional similarity and path length are only interesting in establishing how much spatial

overlap there is in the early and late tasks. We predict that the Learning Position/Goal model will likewise show poor performance in those two metrics, due to its learning of specific locations and goals that only partially overlap from training to testing. We predict that the Learning Behavior Tree model will maintain performance relative to experiment 1, due to its learning relative to the Generate Path behavior. To provide evidence of generality, three data sets are used: the A set, the B set and the two combined (All).

In experiment 3, the training and testing sets are taken from different populations of players performing similar tasks in different map areas. The agents are trained on the A data set and tested on the B data set, then vice-versa. This tests the effectiveness of the Learning Behavior Tree when trained on one set of players in one setting, then used in a different setting with different players. Because the areas are completely different, the Training Set and Learning Position/Goal models cannot work in this experiment. We predict that the Learning Behavior Tree model will continue to maintain performance.

6. Results

Because the two learning models are stochastic, 1000 trials were performed for each experimental result and the means and standard deviations are reported.

The results for experiment 1 are shown in Table 1 for the three data sets A, B and All, and the three agent control models Learning Behavior Tree (Beh. Tree), Learning Position/Goal (Pos/Goal) and Training Set (Train. Set). The metrics are the mean percentage error of distributional similarity between the test and Reference populations (Dist), the difference in average total path length of the test agent population vs. the Reference agent population (PathLen) and the difference in average percent revisits of the test agent population vs. the Reference agent population (Revisit). Smaller absolute magnitudes are better.

For all three data sets, the Learning Behavior Tree model outperforms the other models in the distributional similarity metric. The differences are statistically significant, given the very high number of trials, but seem unlikely to correspond to perceivable differences in crowd similarity. Rather, the results confirm the prediction that the learning models match the baseline for distributional similarity. The Training Set results for path length indicate that different players take widely different paths, so while the learning models underperform in terms of magnitude, the significant overlap suggests that the differences may not be observable. What is clear is that the Learning Behavior Tree model takes more direct paths while the Learning Position/Goal model wanders more. Similarly, the Training Set results for revisits indicate a nearly 10% variance in either direction for human players. However, both learning models revisit substantially less than human players, most likely due to their goal focus.

	Dist	**PathLen**	**Revisit**
Data Set A			
Beh. Tree	**0.32±0.04**	-5.07±3.86	-0.18±0.06
Pos/Goal	0.37±0.06	4.31±3.73	-0.08±0.07
Train. Set	0.35±0.05	**-0.99±4.81**	**0.01±0.09**
Data Set B			
Beh. Tree	**0.30±0.05**	-2.45±3.94	-0.15±0.07
Pos/Goal	0.33±0.07	**0.98±4.15**	-0.09±0.07
Train. Set	0.31±0.07	-1.84±4.6	**0.01±0.09**
Data Set All			
Beh. Tree	**0.27±0.04**	-3.64±2.83	-0.16±0.05
Pos/Goal	0.33±0.04	5.24±3.32	-0.08±0.05
Train. Set	0.33±0.07	**-0.66±3.43**	**0.01±0.07**

Table 6.1: Experiment 1 results, testing and training for each data set randomly divided in half

The results for experiment 2 are shown in Table 6.2, again for the three data sets, the three agent control models and the three metrics. The results of the Training Set model show that the crowd distribution and path lengths are quite dissimilar between the earlier and later halves of the data, while the revisits vary between the A and B sets but are in a similar range to experiment 1. There is, of course, no variation in the Training Set results ($\sigma=0$) since there is only one set of training traces to replay. As predicted, the Learning Position/Goal model performs much more poorly in all metrics. The long path lengths and high positive difference in revisits support the intuitive idea that since the training was only partially applicable, it is randomly wandering a

lot. The results show that the Learning Behavior Tree model maintains its level of performance, and even improves, in this scenario.

	Dist	**PathLen**	**Revisit**
Data Set A			
Beh. Tree	**0.24±0.02**	**-0.93±1.51**	-0.16±0.02
Pos/Goal	0.54±0.02	24.41±1.04	0.33±0.02
Train. Set	0.61±0	-22.1±0	**-0.14±0**
Data Set B			
Beh. Tree	**0.26±0.03**	**2.55±1.4**	**-0.01±0.04**
Pos/Goal	0.75±0.02	16.94±0	0.29±0.02
Train. Set	0.51±0	-11.12±0	0.15±0
Data Set All			
Beh. Tree	**0.21±0.02**	**2.52±1.23**	-0.04±0.02
Pos/Goal	0.71±0.01	32.52±0.64	0.25±0.01
Train. Set	0.56±0	-17.05±0	**0.01±0**

Table 6.2: Experiment 2 results, training for each data set taken longitudinally from the first part of the simulation time and testing from the second part.

The results for experiment 3 are shown in Table 6.3 for the Learning Behavior Tree model training on data set A and testing on data set B and vice versa. The other models have nearly 100% error margins, as expected, and are not shown. These results confirm that the Learning Behavior Tree model maintains its level of performance even across areas and tasks.

Train/Test	**Dist**	**PathLen**	**Revisit**
A/B	0.2±0.01	0.91±1.34	-0.13±0.02
B/A	0.22±0.01	-6.49±1.23	-0.17±0.02

Table 6.3: Experiment 3 results, Learning Behavior Tree trained on data set A/B and tested on the other.

7. Conclusion and Future Work

In this work, we have proposed the use of Learning Behavior Trees to provide context and structure to machine learning for the task of simulating a population of human players in a virtual world. This approach seeks to create flexible agent controllers that combine human authored behaviors, such as path planning and moving through waypoints, with learned behaviors in a declarative control structure. These experiments provide evidence that such a controller can simulate a population of agents to show similarity to human movement behavior in a task-oriented virtual world. Importantly, it maintains performance when testing with increasingly different data from training. However, it does not show as strong similarity by the local metric of revisits. This evidence does not claim to prove that the Behavior Tree is necessary to this model. Clearly, a finite state machine or ad hoc code could be used. Rather, we propose that the advantage to Learning Behavior Trees is the ability to explicitly situate training and execution of learned behaviors within arbitrarily complex, human-authored behaviors. We will next be evaluating the effectiveness of this approach for multiple points of learning within the tree, such as simultaneously learning to simulate travel, decisions regarding which tasks to pursue, and decisions regarding when to interact with other entities. We will also pursue human testing for qualitative evaluation of similarity.

8. Acknowledgements

This material is based upon work supported in part by the U.S. Army Research Laboratory and the U.S. Army Research Office under grant number W911NF-11-1-0150.

9. References

Bainbridge, W.S. (2007). The Scientific Research Potential of Virtual Worlds. *Science.* Vol. 317 no. 5837 pp. 472-476.

Bakkes, S., Spronck, P., and van Lankveld, G. (2012). Player Behavioral Modelling for Video Games. *Entertainment Computing*, Vol. 3, Nr. 3, pp. 71–79.

Best, B., Lebiere, C., & Scarpinatto, C. (2002). A model of synthetic opponents in MOUT training simulations using the ACT-R cognitive architecture. In Proceedings of the Eleventh Conference on Computer Generated Forces and Behavior Representation. Orlando, FL.

Champandard, A. J. (2008). Getting started with decision making and control systems. *AI Game Programming Wisdom 4*. Boston, Massachusetts: Course Technology. 257-263.

Dickey, M.D. (2005). Three-dimensional virtual worlds and distance learning: two case studies of Active Worlds as a medium for distance education. *British Journal of Educational Technology*. Volume 36, Issue 3, pages 439–451.

Galway, L., Charles, D. and Black, M. (2008). Machine learning in digital games: a survey. *Artificial Intelligence Review*. Volume 29, Number 2, 123-161.

Geisler, B. (2004). Integrated machine learning for behavior modeling in video games. In: Fu D, Henke S, Orkin J (eds) *Challenges in game artificial intelligence: papers from the 2004 AAAI workshop*. AAAI Press, Menlo Park, pp 54–62.

Gorski, N. and Laird, J. (2006). Experiments in Transfer Across Multiple Learning Mechanisms. In ICML Workshop on Structural Knowledge Transfer for Machine Learning.

Henry, P., Vollmer, C., Ferris, B. and Fox, D. (2010). Learning to navigate through crowded environments. In *Proceedings ot the 2010 IEEE International Conference on Robotics and Automation*, Anchorage Convention District, May 3-8, 2010, Anchorage, Alaska, USA

Isla, D. (2005). Handling complexity in the Halo 2 AI. In *Proceedings of the GDC 2005.* Gamasutra.

Karpov, I.V., Schrum, J., Miikkulainen, R. (2012). Believable Bot Navigation via Playback of Human Traces. In Philip F. Hingston, editors, *Believable Bots*, 151--170, 2012. Springer Berlin Heidelberg.

Marthi, B., Russell, S., Latham, D. and Guestrin, C. (2005). Concurrent Hierarchical Reinforcement Learning. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland, UK, July 30-August 5.

Schrum, J., Karpov, I.V. and Miikkulain, R. (2012). Humanlike Combat Behavior via Multiobjective Neuroevolution. In Philip F. Hingston, editors, *Believable Bots*, 119--150, 2012. Springer Berlin Heidelberg.

Shao, W., and Terzopoulos, D. (2005). Autonomous pedestrians. In SCA '05: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 19–28.

Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., and Postma, E. (2006). Adaptive Game AI with Dynamic Scripting. *Machine Learning*, Vol. 63, No. 3, pp. 217-248. (Springer DOI: 10.1007/s10994-006-6205-6)

Thurau, C., Bauckhage, C., Sagerer, G. (2003). Combining self-organizing maps and multilayer perceptrons to learn bot-behavior for a commercial game. In Mehdi Q, Gough N, Natkin S (eds) *Proceedings of the 4th international conference on intelligent games and simulation*. Eurosis, pp 119–123.

Togelius, J., Burrow, P. and Lucas, S.M. (2007). Multi-population competitive co-evolution of car racing controllers. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, 4043-4050.

Treuille, A., Cooper, S. and Popović, Z. (2006). Continuum Crowds. *ACM Trans. Graph*, vol 25, pg 1160-1168.

Wray, R., Laird, J., Nuxoll, A., Stokes, D. and Kerfoot, A. (2005). Synthetic Adversaries for Urban Combat Training. *AI Magazine*, Volume 26, Number 3.

Author Biographies

EMMETT TOMAI is an Assistant Professor in the Department of Computer Science at the University of Texas – Pan American. He earned his Ph.D. from Northwestern University in computational understanding of natural language semantics. His research interests include computational storytelling and narrative understanding through intelligent, believable agents in virtual worlds and video games.

ROSENDO SALAZAR is an undergraduate in the Department of Computer Science at The University of Texas-Pan American. He currently is a member of the Center of Excellence in STEM Education Student Research Program.

ROBERTO FLORES is an undergraduate in the Computer Engineering program at the University of Texas Pan-American, and current member of the Center of Excellence in STEM Education Student Research Program.