Developing an Adaptive Opponent for Tactical Training

Jeremy Ludwig[1] and Bart Presnell[1]

[1] Stottler Henke Associates, Inc., San Mateo, CA, USA

Abstract. This paper describes an effort to create adaptive opponents for simulation-based air combat, where the opponents behave realistically while at the same time fulfilling instructional objectives. Three different models are developed to control the behavior of red pilots against (simulated) blue trainees in a set of 2v2 scenarios. These models are then evaluated on their tactical and instructional performance, with the machine-learning model performing on par with the two hand-constructed models. The contribution of this paper is to investigate technology and infrastructure enhancements that could be made to existing systems used for simulation-based air combat training.

Keywords: Behavior Modeling; Simulation; Adaptive Training

1 Introduction

This paper describes an effort to create adaptive opponents for simulation-based training, where the opponents behave realistically while at the same time fulfilling instructional objectives. The domain for this work is air combat, where US Air Force pilots train in a simulator against computer-generated enemies. The effort is one example of a behavior model built under the research program described in [1]. The objective of this program is creating smart and agile opponents that respond more realistically, are less predictable, and take advantage of errors made by trainees while still providing specific types of instructional opportunities.

The remainder of the paper includes an overview of the methods used to create and evaluate the behavior models. Following this, the results section highlights the tactical and instructional intelligence demonstrated in the model. Finally, the discussion section reviews the implications of the results and presents future work.

1.1 Related Work

There is a substantial amount of related work toward creating intelligent, adaptive opponents for games and simulations and in developing systems for autonomous unmanned aerial vehicles. This paper will focus on a small subset of highly related work modelling opponent pilots in the air combat domain.

One of the early, and still active, success stories in this domain is TacAir Soar [2]. This system used production rules created for the Soar architecture. TacAir Soar was used for operational training, flying many different types of fixed-wing aircraft in simulations using appropriate doctrine and tactics. More recent work in this area includes utilizing genetic algorithms [3] and neural networks [4] to best human opponents. Developing a capable adversary in this domain is an issue that has been pursued for quite some time and is still an open problem.

The work presented in [5] applies to finite state machines combined with dynamic scripting to the problem of creating an opponent for air combat training. We followed nearly the same behavior representation approach as [5]–combining hierarchical finite state machines with dynamic scripting.

However, an example of a behavior modelling approach currently in use within the US Air Force is the Next Generation Threat System (NGTS) [6]. NGTS is used across the US government and DoD to develop behavior models that human pilots train against. The contribution of this paper is to investigate future enhancements to systems currently in use, such as the NGTS, that would result in improved training outcomes.

2 Methods

This section includes the methods used both to create and evaluate the behavior models. First, we provide an overview of the SimBionic architecture and dynamic scripting. Following this is a description `of the specific modeling approach that we took using these tools. Finally, we present an overview of the evaluation approach.

2.1 Agent Architecture

This section describes the SimBionic agent architecture, which is followed by a description of the Dynamic Scripting reinforcement learning algorithm. The SimBionic architecture, along with a Dynamic Scripting extension, are available as open source on GitHub [7].

SimBionic. The goal of the SimBionic architecture [8] is to make it possible to specify real-time intelligent software agents quickly, visually, and intuitively by drawing and configuring behavioral transition networks (BTNs) as shown in Figure 1. Each BTN is a network of nodes joined by connectors (links), similar to a flow chart or finite state machine. Visual logic makes it easy to show, discuss, and verify the behaviors with members of the development team, subject matter experts, and other stakeholders. The SimBionic architecture provides three components. The SimBionic Visual IDE application enables modelers to specify intelligent agent behaviors by creating and saving BTNs that are read and executed by the SimBionic Run-time System**.** The run-time software library connects to the simulation API to query for state information and execute actions in the simulation, as specified by the BTNs. The SimBionic Debugger application helps developers test and debug behavior logic by stepping through the execution of the BTNs and inspecting the values of local and global variables.

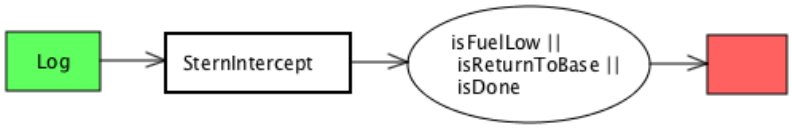

Figure 1. This behavior will attempt to perform a stern intercept on an aircraft until specific conditions are met.

As an example, while controlling an aircraft in the simulation, the Figure 1 BTN is invoked as the initial behavior. Execution of this BTN starts in the green action node (Log) and then transitions to the SternIntercept node—which is a reference to another BTN. At this point, flow of control is passed to the SternIntercept BTN (Figure 2). Execution of the SternIntercept BTN then starts in the green action node and will eventually call other BTNs (SternInterceptLowAA and SternInterceptHighAA). Meanwhile, the initial behavior is monitoring in the background for the conditional node (isFuelLow, isReturnToBase, or isDone) to be true. As soon as one of these becomes true, the SternIntercept BTN is interrupted and control is returned to the initial behavior, which will transition to the red action node (Final).

Each node in a SimBionic BTN includes both a user-friendly description (e.g., "High AA?") as well as JavaScript or Java code that will be called when the node is executed, as shown in the lower left of Figure 2. The overall behavior model includes multiple levels: i) the high-level flow of control represented in the BTNs, ii) the lower-level building blocks written in Java, and iii) the simulation API that carries out actions and provides sensory data.

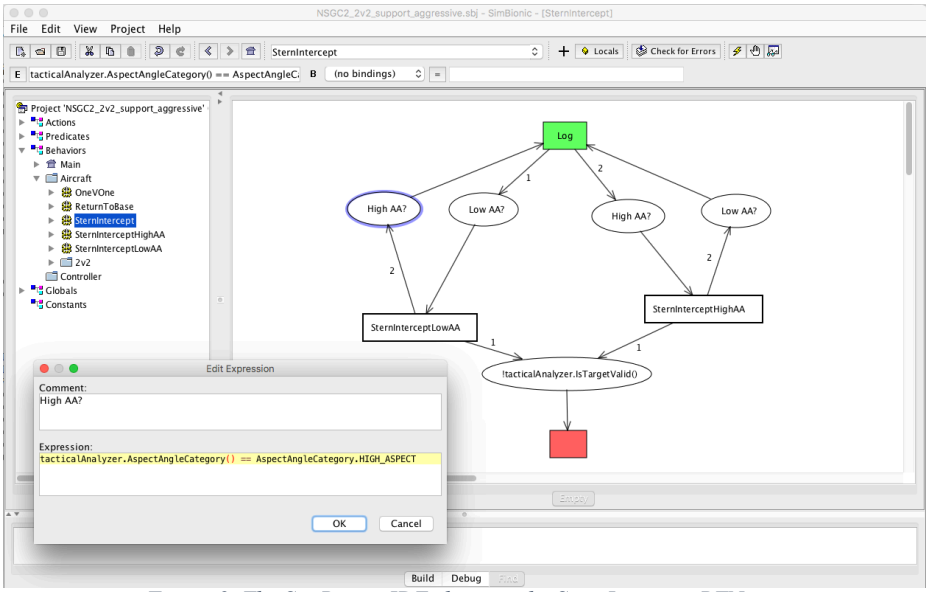

Figure 2. The SimBionic IDE showing the SternIntercept BTN.

A SimBionic entity is a thread of execution. The SimBionic engine can contain one or more entities, each executing its own set of BTNs simultaneously. When using SimBionic to develop agents for a game or simulation, different entities might correspond to simulated characters or computer-generated forces, each BTN acting independently (though they may communicate with each other to coordinate their actions).

Dynamic Scripting. Dynamic scripting [9] is an online reinforcement learning algorithm developed specifically to control the behavior of adversaries in modern computer games and simulations. Put simply, dynamic scripting attempts to learn a subset of IF-THEN rules (called *actions)* that allows the entity to perform well. The subset, chosen from the larger set, is the *script*, in dynamic scripting. Dynamic scripting makes a specific tradeoff for games and simulators, favoring speed of learning over context sensitivity.

More concretely, actions contain i) a value, ii) an optional IF clause that describes when an action can be applied based on the perceived game state, and iii) a user-defined priority that captures domain knowledge about the relative importance of each action. Action values are used to create scripts of length *n* prior to a scenario by selecting rules in a value-proportionate manner (e.g., softmax) from the complete set of actions available to the agent. During a scenario, *applicable* actions are selected from the script in priority order first, the action value second. *Applicability* is determined by the perceived game state and the action's IF clause. At the end of a scenario, action values are updated using the dynamic scripting updating function combined with a domain-specific reward function created by the behavior author. The reward is distributed primarily to the actions selected in the episode and then to actions in the script that were not selected, with a smaller negative reward given to actions not included in the script.

There are two primary benefits of this learning approach in the context of creating opponents in the air combat domain [5]. First, the algorithm can learn quickly and continuously—a few scenarios is enough to drive an obvious change in behavior. Second, the algorithm learns within a well-bounded space. Instructor pilots develop the space of possible actions, and the algorithm searches to find a set of actions from this space that work well together. This leads to behavior that is adaptive from the trainee's perspective while still being predictable from the instructor's perspective.

2.2 Modeling Approach

We investigated three different models to control the behavior of red pilots in a set of scenarios that pit two red aircraft against two blue (2v2) as shown in Figure 3. All three models were built in the SimBionic modeling architecture. Following the discussion of the three models is a description of some common functionality.

- **SimBionic.** A handcrafted behavior model using SimBionic's hierarchical behavior transition network representation.
- **Rules.** The second model is simply composed of a hand-selected set of IF-THEN rules, modeled in a SimBionic BTN.
- **Dynamic Scripting (DS).** The third model uses reinforcement learning to learn the behavior model from experience from among a larger set of IF-THEN rules.

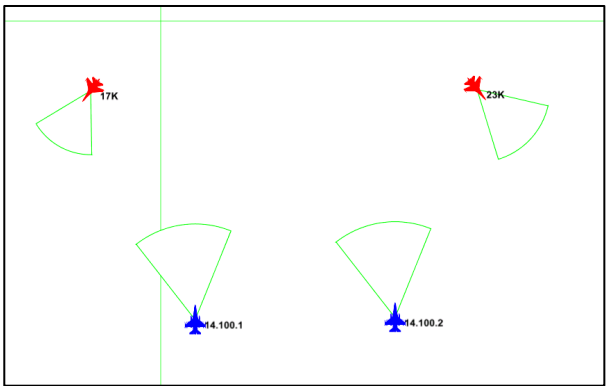

Figure 3. Example 2v2 scenario.

SimBionic. For the 2v2 behavior model, we developed three specific SimBionic entities. The first and second entities each control one of the two aircraft. These two entities run their own versions of the same set of hierarchical BTNs, similar to that shown in Figure 1 and Figure 2. The third entity is a controller, which assigns targets to the red aircraft. This entity uses a different set of BTNs. The controller BTN is the only explicit instance of coordination that happens between the two aircraft—all other coordination is implicit in either the BTNs or the low-level Java building blocks.

Rules. After developing the SimBionic 2v2 model using hierarchical BTNs, a second, much simpler model to control the red aircraft was created. This model was composed of six IF-THEN rules implemented in a single BTN in SimBionic. The Rules model also reused the controller entity from the SimBionic model to assign targets to the two red aircraft. This enabled us to take advantage of all of the integration work and building blocks that had already been completed and quickly create the rule-based model. The idea behind this second model was to explore the performance of an extremely simple approach relative to the hierarchical model.

Dynamic Scripting. For the 2v2 Dynamic Scripting behavior model, thirteen possible actions to select from were created. These included the six actions from the Rules model and seven new actions. Since each DS action is an IF-THEN rule, we represented all thirteen rules in a single SimBionic BTN. The result is a BTN that looks very much like the Rules model, just with more nodes and transitions. The primary difference is the introduction of a special type of dynamic scripting node that indicates dynamic scripting transition selection should be used rather than the standard SimBionic transitions. When executing in a scenario, the first time a chooseDS node is reached, a script will be generated based on action weights. Subsequent transition through the BTN will use the same script. There will be a corresponding rewardDS node to apply the reward function at the end of the scenario, which will update the weights and reset Dynamic Scripting so that a new script is selected next time the chooseDS node is reached. With this setup, each red aircraft learns its own script.

The rewards are given to the behaviors based on two events. The first event is the occurrence of any frame in which a red entity controlled by a behavior model is threatening a blue entity. For this event, a small reward is evenly shared between each red entity that is targeting the threatened blue entity. The intention is to provide a reward as entities move into position or distract the target. The second reward event occurs when a red entity is removed from the simulation. This reward takes the form of a large negative reward that is applied only to the red entity that was captured. This decision was based on the assumption that individual behaviors are responsible for maneuvering the entity into a dangerous position, regardless of how other entities are behaving.

Prior to evaluation, the dynamic scripting model was trained by running it against twelve scenarios five times each, for a total of 60 training runs. The behavior model was then frozen for evaluation. Learning was disabled during the evaluation so as to generate more stable results.

Common Functionality. There is significant common functionality across all three models. This includes being modeled in the SimBionic architecture, sharing the same Controller BTN for target assignments, and sharing the same lower-level building blocks developed in Java that execute actions such as turns or attempts to intercept another aircraft.

These three models also share constrained variability through the use of these thresholds in the SimBionic BTNs and Java building blocks. The thresholds allow behaviors to vary their performance in subtle ways in each scenario. For example, rather than always performing an action at 10 NM, a model using a threshold might perform the action at 9.2 NM one time and 10.7 NM another time. This approach has two advantages. First, the exact behavior of the red aircraft is more difficult to predict. Second, the behavior of the red aircraft is easy to change. Updating the default, min, and max thresholds will affect how the model performs but requires no additional changes to the model.

2.3 Evaluation Approach

To evaluate the tactical intelligence of the three behavior models (SimBionic, Rules, and DS), we ran each model against a set of thirteen scenarios and then averaged the quantitative results. The scenarios are not deterministic and returned slightly different results on different runs, so each scenario was completed twice. We also recorded videos of the evaluation scenarios for qualitative assessment of tactical and instructional intelligence by subject matter experts.

In these scenarios, the behavior models controlled the red aircraft, while the blue aircraft were controlled by simplistic scripts. The quantitative goal was to maximize the number of blue aircraft removed from the scenario (captured) while minimizing the number of red aircraft lost. The qualitative goal was to perform realistically while following an instructor-developed plan that describes at a high level how the red aircraft should act in these scenarios against the blue trainees.

3 Results

The results section highlights the tactical and instructional intelligence demonstrated by the three different behavior models. Tactical intelligence focuses on successfully completing the scenario against a simulated trainee. Instructional intelligence focuses on aspects such as performing more realistically, taking better advantage of mistakes made by the simulated trainee, and being less predictable, than current agent models.

The quantitative results in Table 1 show the aircraft "captured to lost" ratio across the evaluation scenarios. Following the table are three charts showing the average captured and lost aircraft per scenario for each of the three models in each of the evaluation scenarios (Figure 4, Figure 5, & Figure 6).

Table 1. Average model performance across two runs of all evaluation scenarios.

Model	Average # Blue Aircraft Captured [0-2]	Average # Red Aircraft Lost [0-2]	Captured / Lost Ratio
SimBionic	1.54	0.15	10.3
Rules	1.46	0.12	12.2
DS	1.38	0.12	11.5

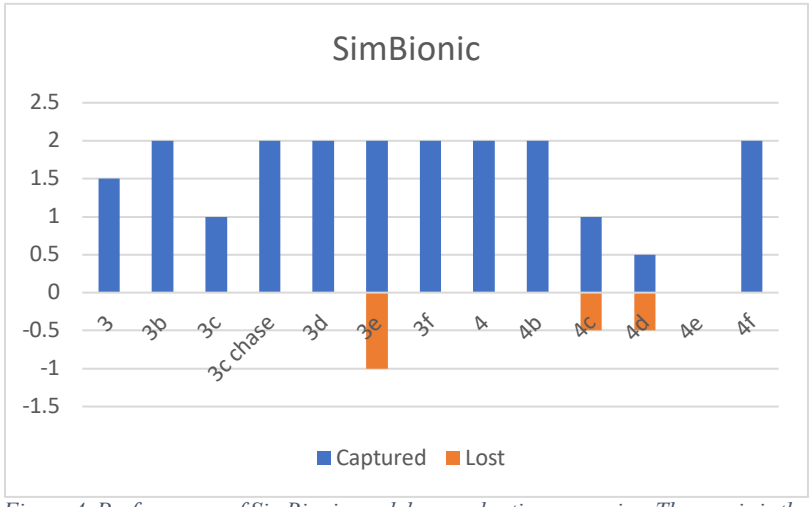

Figure 4. Performance of SimBionic model on evaluation scenarios. The x-axis is the scenario and the y-axis is the number of aircraft captured (blue) and number of aircraft lost (orange).

Figure 5. Performance of Rules model on evaluation scenarios. The x-axis is the scenario and the y-axis is the number of aircraft captured (blue) and number of aircraft lost (orange).

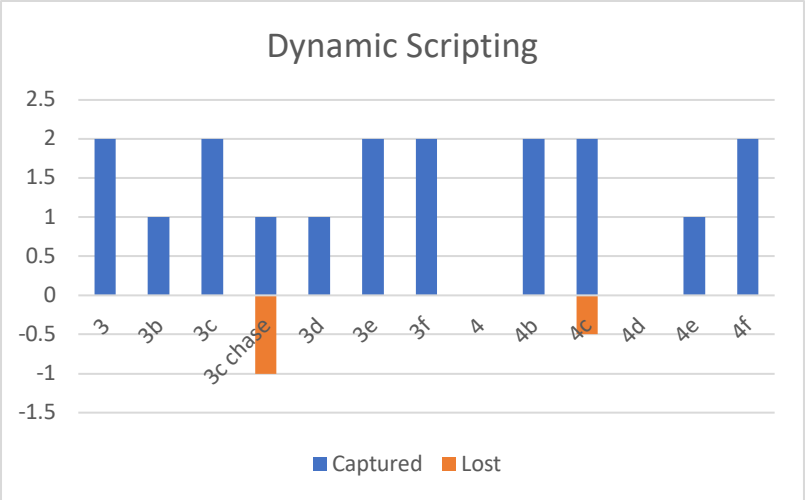

Figure 6. Performance of Dynamic Scripting model on evaluation scenarios. The x-axis is the scenario and the y-axis is the number of aircraft captured (blue) and number of aircraft lost (orange).

Qualitative examination was performed only on the SimBionic scenarios due to the limited amount of subject matter expert time available. The subject matter experts found that the SimBionic model generally behaved realistically and took advantage of trainees' mistakes—with a couple of exceptions. One exception had to do with how red carried out a maneuver. In this case, the error was in the low-level Java code that carries out the maneuver, not with the behavior model itself. This means that all three models

would make the same mistake. The second exception had to do with teamwork, where one red aircraft was not supporting the other correctly under certain circumstances. As this error was embedded in the SimBionic model, the incorrect behavior would not necessarily be present in the Rules or DS models.

4 Discussion

These quantitative results show that all of the models were fairly successful in capturing opposing forces while not getting captured themselves. Using this as the primary metric, the results demonstrate good performance for all three behavior models. What is very surprising is that the simplest behavior model, Rules, displays the best performance in terms of captured-to-lost ratio. The model created via machine learning, DS, performs almost as well. This is not necessarily surprising since DS has access to all of the IF-THEN statements in the Rules model, but it is a good outcome nonetheless. SimBionic performs reasonably well, too—it is just not quite as good as these other models at maximizing the captured-to-lost ratio. It is also interesting to note that the different models have different strengths and weaknesses—e.g., SimBionic was at a stalemate in 4e, where both Rules and DS were able to come out ahead in this scenario.

Instructional intelligence is evaluated relative to the requirements set by the instructor pilots with respect to improved red behavior models: perform realistically, take advantage of mistakes made by trainees, and be less predictable. First, our belief is that the SimBionic model will behave more realistically than the Rules or DS models. Based on personal observation as modelers, we can see that the Rules and DS models behave differently than the more highly constrained SimBionic model. The realism of Rules and DS models has not been independently examined. Second, all three models seem to take advantage of blue mistakes while protecting themselves, given the captured-to-lost ratio. Third, the constrained variability discussed in Common Functionality provides significant variation in aircraft behavior across scenario runs—though this was not directly noticeable to the subject matter experts reviewing model performance.

5 Conclusion

The overall objective of this work is to advance the intersection of cognitive modeling and machine learning in order to develop behavior modeling technology and supporting infrastructure for the US Air Force. Working towards the overall objective, this paper describes the development and evaluation of three different behavior models from both a tactical and instructional perspective. All three models performed well from both perspectives, with the machine learning model performing on par with the other two models after relatively little training.

In future work, we plan to combine the SimBionic and Dynamic Scripting approaches, adding dynamic scripting nodes at various points within the hierarchical SimBionic BTNs. We believe this will best support realistic behavior and adherence to an instructional plan, while at the same time applying machine learning to adapt to the trainees' performance.

Acknowledgements

This article is based upon work supported by the United States Air Force Research Laboratory, Warfighter Readiness Research Division 711 HPW/RHA, under Contract FA8650-16-C-6698. This article is cleared for public release on 14 Dec 2018, Case 88ABW-2018-6265.

Distribution Statement A: Approved for Public Release, Distribution Unlimited.

References

[1] J. Freeman, E. Watz, and W. Bennett, "Adaptive Agents for Adaptive Tactical Training: The State of the Art and Emerging Requirements," to be presented at the HCI INTERNATIONAL 2019, Orlando, FL, US, 2019.

[2] R. M. Jones, J. E. Laird, P. E. Nielsen, K. J. Coulter, P. Kenny, and F. V. Koss, "Automated Intelligent Pilots for Combat Flight Simulation," *AI Mag.*, vol. 20, no. 1, pp. 27–27, Mar. 1999.

[3] N. Ernest, D. Carroll, C. Schumacher, M. Clark, K. Cohen, and G. Lee, "Genetic Fuzzy based Artificial Intelligence for Unmanned Combat Aerial Vehicle Control in Simulated Air Combat Missions," *J Def Manag*, vol. 6, no. 144.

[4] T. Teng, A. Tan, Y. Tan, and A. Yeo, "Self-organizing neural networks for learning air combat maneuvers," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, 2012, pp. 1–8.

[5] A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat, and H. J. van den Herik, "Rapid Adaptation of Air Combat Behaviour," in *ECAI*, 2016.

[6] "NEXT GENERATION THREAT SYSTEM (NGTS)," 10-Dec-2018. [Online]. Available: http://www.navair.navy.mil/nawctsd/pdf/2018-NGTS.pdf.

[7] "SimBionic," 04-Jun-2018. [Online]. Available: https://github.com/StottlerHenkeAssociates/SimBionic. [Accessed: 11-Dec-2018].

[8] D. Fu and R. Houlette, "Putting AI in entertainment: an AI authoring tool for simulation and games," *IEEE Intell. Syst.*, vol. 17, no. 4, pp. 81–84, Jul. 2002.

[9] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma, "Adaptive game AI with dynamic scripting," *Mach. Learn.*, vol. 63, no. 3, pp. 217–248, Jun. 2006.