Autonomous Aircraft Sequencing and Separation with Hierarchical Deep Reinforcement Learning

Marc Brittain
Aerospace Engineering Department
Iowa State University
Ames, IA, USA
mwb@iastate.edu

Peng Wei
Aerospace Engineering Department
Iowa State University
Ames, IA, USA
pwei@iastate.edu

***Abstract*—With the increasing air traffic density and complexity in traditional controlled airspace, and the envisioned large volume vertical takeoff and landing (VTOL) operations in low-altitude airspace for personal air mobility or on-demand air taxi, an autonomous air traffic control system (a fully automated airspace) is needed as the ultimate solution to handle dense, complex and dynamic air traffic in the future. In this work, we design and build an artificial intelligence (AI) agent to perform air traffic control sequencing and separation. The approach is to formulate this problem as a reinforcement learning model and solve it using the hierarchical deep reinforcement learning algorithms. For demonstration, the NASA Sector 33 app has been used as our simulator and learning environment for the agent. Results show that this AI agent can guide aircraft safely and efficiently through "Sector 33" and achieve required separation at the metering fix.**

Keywords: Artificial Intelligence, Autonomous Air Traffic Control, Hierarchical Deep Reinforcement Learning

I. INTRODUCTION

A. Motivation

The original proposal of an autonomous air traffic control system was from Heinz Erzberger and his NASA colleagues, where they believe that a fully automated system, referred as the *Advanced Airspace Concept (AAC)*, is the ultimate solution to accommodate dense, complex and dynamic air traffic in the controlled airspace in the future. The core element of the AAC is called the *Autoresolver*. It was designed to detect and resolve conflicts in en route and terminal airspace. In their papers [1, 2, 3], such an autonomous air traffic control system for automated sequencing and separation is expected to augment human air traffic controllers to increase airspace capacity and enhance operation safety.

In the recent proposals for low-altitude airspace operations, including *UAS Traffic Management (UTM)* [4, 5, 6] for remote-piloted or unmanned autonomous drone operations and *Urban Air Mobility (UAM)* [7, 8, 9, 10] for vertical takeoff and landing (VTOL) personal air travel or urban air taxi operations, an autonomous air traffic control system is needed to communicate with future intelligent aircraft, facilitate on-board autonomy or human operator decisions, and cope with envisioned high-density and on-demand air traffic by providing automated sequencing and separation advisories.

The inspiration of this paper is twofold. First, the authors were amazed by the fact that an artificial intelligence agent called *AlphaGo* built by DeepMind defeated the world champion Ke Jie in three matches of Go in May 2017 [11]. This notable advance in AI field demonstrated the theoretical foundation and computational capability to potentially augment and facilitate human tasks with intelligent agents and AI technologies. Therefore, the authors want to apply the most recent AI frameworks and algorithms to re-visit the autonomous air traffic control idea. Second, the authors were granted the software called *NASA Sector 33* for education and outreach purposes [12]. It is an air traffic control game designed to interest students in air transportation and aviation related careers. It contains 35 problems featuring two to five aircraft. The player needs to apply speed and route controls over these aircraft to satisfy sequencing and separation requirements, as well as, minimize delay. The authors decide to use this software as the game environment and simulator for performance evaluation of our proposed framework and algorithms.

In this paper, a hierarchical deep reinforcement learning framework is proposed to enable autonomous air traffic sequencing and separation, where the input of this agent is the air traffic controller's screen. Through the computer vision techniques, the agent will "see" the screen, comprehend the air traffic situation, and perform online sequential decision making to select actions including speed and route option for each aircraft in real time with ground-based computation. Unlike Erzberger's approach, in our paper the reinforcement learning agent integrates conflict detection and conflict resolution together via a deep reinforcement learning approach. Like Erzberger, we also assume that the computed speed and route advisories will be sent from ground to aircraft via data link. The series of actions will guide the aircraft quickly through "Sector 33" and maintain required safe separation. Our proposed framework and algorithms provide another promising potential solution to enable autonomous air traffic control system.

B. Related Work

There have been many important contributions to the topic of autonomous air traffic control. One of the most promising and well-known lines of work is the Autoresolver designed and developed by Heinz Erzberger and his NASA colleagues [1, 2, 3]. It employs an iterative approach, sequentially computing and evaluating candidate trajectories, until a trajectory is found that satisfies all of the resolution conditions. The candidate trajectory is then output by the algorithm as the conflict resolution trajectory. The Autoresolver is a physics based approach that involves separate components of conflict detection and conflict resolution. It has been tested in various large-scale simulation scenarios. In addition, the Autoresolver is being verified and validated by NASA researchers using formal methods.

Reinforcement learning and deep Q-networks have been demonstrated to play games such as Go, Atari and Warcraft [13, 14, 15]. The results from these papers show that a well-designed, sophisticated AI agent is capable of performing high-level tasks, as well as, learning complex strategies. Therefore, we are encouraged to apply the reinforcement learning framework to solve an air traffic control game and set up an environment for the AI agent to learn the fundamental air traffic control tasks, i.e., aircraft sequencing and separation.

In this paper, the reinforcement learning framework and a novel hierarchical deep agent algorithm are developed to solve the sequencing and separation problem with delay minimization for autonomous air traffic control. The results show that the algorithm has very promising performance.

The structure of this paper is as follows: in Section II, the background of reinforcement learning and deep Q-network will be introduced. In Section III, the description of the problem and its mathematical formulation of reinforcement learning are presented. Section IV presents our designed hierarchical deep agent algorithm to solve this problem. The numerical experiment and results are shown in Section V. And Section VI concludes this paper.

II. Background

A. Reinforcement Learning

Reinforcement learning is one type of sequential decision making where the goal is to learn how to act optimally in a given environment with unknown dynamics. A reinforcement learning problem involves an environment, an agent, and different actions the agent can take in this environment. The agent is unique to the environment and we assume the agent is only interacting with one environment. Let t represent the current time, then the components that make up a reinforcement learning problem are as follows:

- S - The state space S is a set of all possible states in the environment
- A - The action space A is a set of all actions the agent can take in the environment
- $r(s_t, a_t, s_{t+1})$ - The reward function determines how much reward the agent is able to acquire for a given (s_t, a_t, s_{t+1}) transition
- $\gamma \in [0, 1]$ - A discount factor determines how far in the future to look for rewards. As $\gamma \rightarrow 0$, only immediate rewards are considered, whereas, when $\gamma \rightarrow 1$, future rewards are getting prioritized.

S contains all information about the environment and each element s_{t+1} can be considered a snapshot of the environment at time t. The agent accepts s_t and with this, the agent then decides an action, a_t. By taking action a_t, the state is now updated to s_{t+1} and there is an associated reward from making the transition from $s_t \rightarrow s_{t+1}$. How the state evolves from $s_t \rightarrow s_{t+1}$ given action a_t is dependent upon the dynamics of the system, which is unknown. The reward function is user defined, but needs to be carefully designed to reflect the goal of the agent. Fig. 1 shows the progression of a reinforcement learning problem.

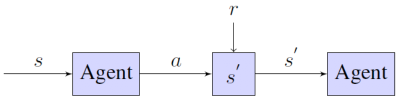

Figure 1. Progression of a reinforcement learning problem within an environment.

From this framework, the agent is able to learn the optimal decisions in each state of the environment by maximizing a cumulative reward function. We call the sequential actions the agent makes in the environment a *policy*. Let π represent some policy and T represent the total time for a given environment, then the *optimal policy* can be defined as:

$$\pi^* = \arg\max_\pi \mathrm{E}[\sum_{t=0}^{T}(r(s_t, a_t, s_{t+1}) \mid \pi)]. \qquad (1)$$

If we define the reward for actions we deem “optimal” very high, then by maximizing the total reward, we have found the optimal solution to the problem.

B. Q-Learning

One of the most fundamental reinforcement learning algorithms is known as Q-learning. This popular learning algorithm was introduced by Watkins and the goal is to maximize a cumulative reward by selecting an appropriate action in each state [16]. The idea of Q-learning is to estimate a value Q for each state and action pair (s, a) in an environment that directly reflects the future reward associated with taking such an action in this state. By doing this, we can extract the policy that reflects the optimal actions for an agent to take. The policy can be thought of as a mapping or a look-up table, where at each state, the policy tells the agent which action is the best one to take. During each learning iteration, the Q-values are updated as follows:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r+\gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)). \quad (2)$$

In (2), α represents the learning rate, r represents the reward for a given state and action, and γ represents the discount factor. One can see that in the max $Q(s_{t+1}, a_{t+1})$ term, the idea is to determine the best possible future reward by taking this action.

C. Deep Q-Network (DQN)

While Q-learning performs well in environments where the state-space is small, as the state-space begins to increase, Q-learning becomes intractable. It is because there is now a need for more experience (more game episodes to be played) in the environment to allow convergence of the Q-values. To obtain Q-value estimates in environments where the state-space is large, the agent must now generalize from limited experience to states that may have not been visited [17]. One of the most widely used function approximation techniques for Q-learning is deep Q-networks (DQN), which involves using a neural network to approximate the Q-values for all the states. With standard Q-learning, the Q-value was a function of *Q(s, a)*, but with DQN the Q-value is now a function of *Q(s, a, θ)*, where θ is the parameters of the neural network. Given an n-dimensional state-space with an m-dimensional action space, the neural network creates a map from $\Re^n \rightarrow \Re^m$. As mentioned by Van Hasselt et al., incorporating a target network and experience replay are the two main ingredients for DQN [18]. The target network with parameters θ^-, is equivalent to the online network, but the weights (θ^-) are updated every τ time steps. The target used by DQN can then be written as:

$$Y_t^{DQN} = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1};\theta_t^-). \quad (3)$$

The idea of experience replay is that for a certain amount of time, observed transitions are stored and then sampled uniformly to update the network. By incorporating the target network, as well as, experience replay, this can drastically improve the performance of the algorithm [18].

D. Double Deep Q-Network (DDQN)

In Q-learning and DQN there is the use of a max operator to select which action results in the largest potential future reward. Van Hasselt et al. showed that due to this max operation, the network is more likely to overestimate the values, resulting in overoptimistic Q-value estimations [19]. The idea introduced by Hasselt was to decouple the max operation to prevent this overestimation to create what is called double deep Q-network (DDQN) [20]. To decouple the max operator, a second value function must be introduced, including a second network with weights θ'. During each training iteration, one set of weights determines the greedy policy and the other then determine the Q-value associated. Formulating (3) as a DDQN problem:

$$Y_t^{DDQN} = r_{t+1} + \gamma Q(s_{t+1}, \arg\max_{a_{t+1}} Q(s_{t+1},a_{t+1};\theta_t);\theta_t'). \quad (4)$$

In (4), it can be seen that the max operator has been removed and we are now including an argmax function to determine the best action due to the online weights. We then use that action, along with the second set of weights to determine the estimated Q-value.

III. PROBLEM FORMULATION

A. Problem Statement

In real world practice, the air traffic controllers in en route and terminal sectors are responsible for sequencing and separating aircraft. In our research, we used the NASA Sector 33 web-based application (a video game) as our air traffic control simulator. The application has a set of problems whose solutions are difficult to find. To evaluate the performance of our hierarchical deep agent algorithm, we selected three different game scenarios with various difficulty levels and constraints.

1) Objective: The objective in the NASA Sector 33 game environment is to maintain a safe separation between aircraft, resolve conflict, and minimize delay by making appropriate route changes and providing speed advisories. Ultimately, we want to guide each aircraft quickly through the metering fix. Unlike the real-world air traffic control scenario, in this game environment there is also a unique "final metered position" right after the metering fix for each aircraft. It is not too difficult for a good human player to obtain a *feasible solution* in the game, where the aircraft are close to their final metered position, safe separation is maintained, and conflicts are resolved. In order to obtain the *optimal solution* in this environment, the aircraft have to maintain safe separation, resolve conflict, and arrive at their final metered position with no delay. However, obtaining the optimal solution in this environment is much more difficult than obtaining the feasible solution, due to the fact that each aircraft has to follow the optimal speed and route at every time-step. If one speed change is made incorrectly, then the optimal solution will not be achieved.

2) Constraints: There are many constraints in the NASA Sector 33 game environment to help resemble a real-world air traffic environment. For each problem, there is a fixed number of aircraft. For some problems, there is only two aircraft, some there is three aircraft, and this number increased to a maximum of five aircraft. The next constraint imposed a limit on the number of route changes for a given aircraft. For example, one problem might allow for a single aircraft to change routes, while another problem would allow both aircraft to change routes, thus increasing the complexity. Weather also imposed an additional limit on the number of aircraft route changes. In some of the problems, there is a storm blocking one of the routes, so now there is no option to select the corresponding route anymore. One of the strictest constraints in the game is time. In each problem, there is a timer for how long the episode lasts and there is only enough time for the optimal solution to be obtained. This meant that to obtain the optimal solution, the aircraft has to be at their final metered position when the timer is at 0:00, otherwise it is not an optimal solution. The last constraint in the game is the individual speed of each aircraft. Each aircraft has the ability to fly at six different speeds ranging from 300 knots to 600 knots.

B. *Reinforcement Learning Formulation*

Here we formulate the NASA Sector 33 environment as a reinforcement learning problem and define the state-space, action-space, and reward functions for the parent agent, as well as, the child agent.

1) State Space: A state contains all the information the AI agent needs to make decisions. The state information was composed of different information for the parent agent and child agent. For the parent agent, the information included in the state was a screen-shot of the game screen. For the child agent, if we let i represent a given aircraft, then the information included in the state was: aircraft positions (x_i, y_i), aircraft speeds v_i, and route information. Route information included the combination of routes for both aircraft, defined as C_j, where j represents a given route combination. From this, we can see that the state-space for the parent agent is constant, since it only depends on the number of pixels in the screen-shot. Suppose there are $m \times m$ pixels in the screen-shot and n number of aircraft, then the state-space can be represented with $m \times m$ numbers for the parent agent and $2 \times n + n + 1$ for the child agent. Fig. 2 shows an example of a state in the NASA Sector 33 game environment.

If we consider Fig. 2 as an example, we can acquire all of the state information we need from the game-screen. For the parent agent, the state will be represented as follows:

$$S_P = (p_1, p_2, p_3, \ldots, p_{m \times m}),$$

where p_k represents the intensity tuple of pixel k. For each pixel in the game screen, there is an associated pixel intensity tuple (red, green, blue) that contains the intensity for each color ranging from [0, 255]. For example, the color green is represented as (0, 255, 0), the color blue is represented as (0, 0, 255) and the color red is represented as (255, 0, 0). For the child agent, the state is defined as:

$$S_C = (x_1, y_1, x_2, y_2, v_1, v_2, C_j),$$

where the subscript represents a specific aircraft and j is the combination of route that the aircraft will take.

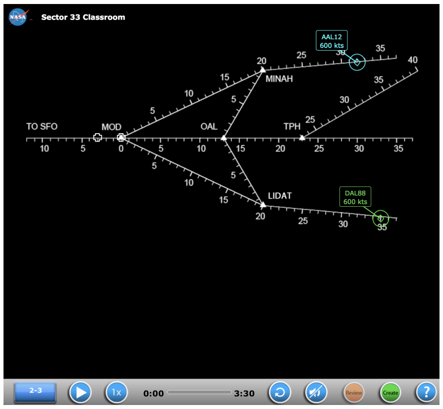

Figure 2. Example of a state in the NASA Sector 33 game environment.

2) Action Space: At each time-step, the parent agent and child agent can make a decision to change the route of the aircraft and change the speeds of the aircraft, respectively. The only difference is the decision time-step for the parent agent and child agent. The parent agent takes one action every episode, where an episode is defined as an entire play through the game. The child agent takes one action every four seconds within the episode to provide more control over the aircraft once the route combination is determined. The action-space for the parent agent can be defined as follows:

$$A_P = (C_1, \ldots, C_j) \;\; \forall \; j,$$

where j is the number of route combinations for the aircraft. If we consider the example in Fig. 2, the action-space for the parent agent will be:

$$A_C = (C_1, C_2, C_3, C_4).$$

This is because each aircraft can take two unique routes, which equates to four unique route combinations.

For the child agent, the action-space is defined as:

$$A_C = (U_1, \ldots, U_k) \;\; \forall \; k,$$

where we define U as all of the possible combinations of speeds for the aircraft and k as a unique speed combination.

3) Terminal State: Termination in the episode could be achieved in three different ways:

- Goal reached (optimal) - All aircraft made it to their final metered position, maintained safe separation, and avoided collision. This meant that:

$$| g_{x_i} - x_i | = 0, \ \forall i$$

- Out of time (feasible) - The aircraft did not arrive at their final metered position, but might have given more time.

$$| g_{x_i} - x_i | > 0, \ \forall i$$

- Collision - The aircraft collided with one another.

$$\sqrt{(y_j - y_i)^2 + (x_j - x_i)^2} < \delta, \forall\, i \neq j$$

where δ is in terms of the pixel distance, and g_x is defined as the set of goal positions for each aircraft.

By observing the current state, we were able to see if any of the terminal states were obtained. For example, let the current state be defined as (x_1, y_1, x_2, y_2, v_1, v_2, C_j), then if

$$| g_{x_1} - x_1 | + | g_{x_2} - x_2 | = 0,$$

we have obtained the optimal solution to the problem.

4) Reward Function: The reward function for the parent agent and child agent needed to be designed to reflect the goal of this paper: safe separation, minimizing delay of arriving at final metered position, and choosing the optimal route combination. We were able to capture our goals in the following reward functions for the parent agent and child agent:

Parent agent

$$r_P = \frac{1}{\sum_{i=1}^{N} | g_{x_i} - x_i |}$$

Child agent

$$r_C = 0.001 \sum_{i=1}^{N} v_i - 0.6,$$

where N is the number of aircraft. In the reward for the child agent, we included two constants (0.001 and -0.6). The reason for adding the factor of 0.001 is to scale the rewards between [-1, 1]. The addition of 0.6 is to penalize slower aircraft speeds and to reward faster aircraft speeds. These rewards were obtained at each time-step for the parent and child agent. If a terminal state was reached, then there was an additional reward that was added for each scenario: -10 for collision, -3 for out of time, and +10 for optimal solution. With these reward functions, the AI agent will prioritize choosing the route combination that allows the aircraft to arrive as quickly as possible, as well as, choosing the fastest speed for the aircraft without creating any conflict.

IV. Solution Approach

To solve the NASA Sector 33 environment, we designed and developed a novel reinforcement learning algorithm called the hierarchical deep agent. In this section, we introduce and describe the algorithm, then we explain why this algorithm is needed to solve this game.

A. Hierarchical Deep Agent

To formulate this environment as a reinforcement learning problem, we found that we were unable to formulate this environment as a typical single agent environment due to the non-Markovian property that this problem involves. The route change for an aircraft can only happen during a small window of time, therefore, if formulated as a single agent problem, the action of changing routes would not be chosen nearly as often leading to a very slow convergence time.

To solve this problem, we used a parent agent who has a second agent *nested* within. The parent agent will take an action (changing route) and then the child agent will control the actions of changing speeds. One of the important differences between the parent and child agents is the state-space. The state-space for the parent agent can be represented as screen pixels or in terms of the aircraft positions and speeds. The child agent has the same state as the parent agent if we use the aircraft positions and speeds, but we also add another dimension to the child agent state. This new dimension we add is the action of the parent agent.

We do this to decouple the action sets of changing route and changing speed. Then we can have the parent agent take one action in the beginning of the episode, followed by the child agent adding this action to its state and proceeding with the actions of changing speed. Fig. 3 provides a diagram of the progression of information from the parent agent to the child agent. We can see the initial state s_P is input into the parent agent, then the parent agent takes action a. From there, action a is now included into the child agent state, s_C. From there, the child agent is able to make all successive decisions with this information. Algorithm 1 provides a pseudocode for formulating a hierarchical deep reinforcement learning problem.

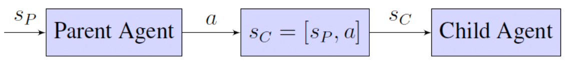

Figure 3. Progression from Parent agent to Child agent.

```
Algorithm 1 Hierarchical Deep Agent
  Initialize: Parent Agent
  Initialize: Child Agent
  Initialize: $s_P$
  reward = 0
  number of episodes = $n$
  for $i = 1$ to $n$ do
    $a_P = ChooseAction$(ParentAgent)
    $s_C = [s_P\ ,\ a_C]$
    repeat
      $a_c = ChooseAction$(ChildAgent)
      $s', r_C = SimulateEnvironment$
      receive parent agent reward = $r_P\ (s')$
      $reward = reward + r$
      $update$(ChildAgent)
    until Terminal
    $update$(ParentAgent)
  end for
```

B. Overall Approach

In our formulation of the hierarchical deep agent, we decided to use the game screen (raw pixels) as input for the parent agent. By using the game screen, the parent agent can "see" and "comprehend" the air traffic situation by abstracting the important features through the hidden layers of double deep Q-network (DDQN), such as relative aircraft positions without explicitly providing the information. Then the parent agent is able to make route selections for all aircraft at the output layer. In this way, the parent agent DDQN integrates the conflict detection task ("seeing" and "comprehension") and first part of conflict resolution task of route selection ("decision making"). Fig. 4 shows an illustrative example of the parent agent architecture. In Fig. 4, the "hidden layer" is a simplified illustration to represent the entire DDQN. In this specific game scenario shown in Fig. 4, we have two aircraft to control. The upper one has two route options, and the lower one has only one route option. Therefore, the total route combinations for these two aircraft is two.

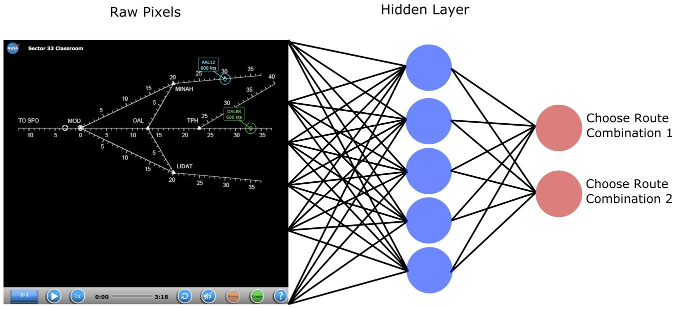

Figure 4. Illustrative example of the decision-making process for the parent agent.

With the route combination selected by the parent agent, the child agent will include this information to its current state and proceed with learning how to compute the speed adjustment advisories. Fig. 5 illustrates a specific example for the child agent. Here we define the hidden layer as the entire DDQN representation introduced, and the output as the speed controls the child agent can implement. In our case study, the output layer is the speed combinations of the aircraft.

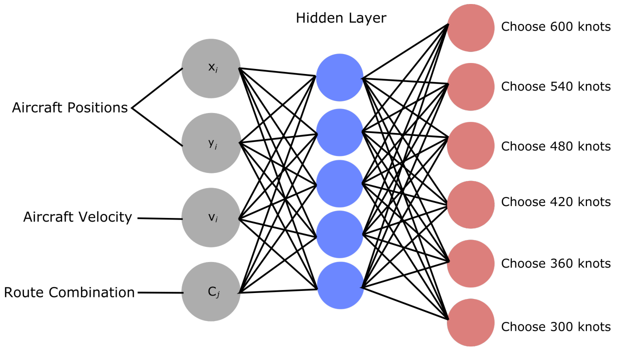

Figure 5. Illustrative example of the decision-making process for the child agent.

V. Numerical Experiments

A. Interface

To test the performance of our proposed algorithm, an interface was built between the web-based NASA Sector 33 application and our code so that the AI agent can operate all functions in the game. The state-space information for the agent was obtained using computer vision techniques. By capturing screen-shots of the game screen every four seconds, we could then extract the aircraft position in terms of the (x_i, y_i) pixel location. By game design, when restarting a particular problem, all objectives were the same: final aircraft metered positions, aircraft initial position, and available speed changes did not change. We also included an additional feature that is not available by default in the web-based version of NASA Sector 33. This feature is to terminate the game if two aircraft are within in a 30-pixel radius (aircraft collision).

B. Environment Setting

For each problem in NASA Sector 33, we discretized the environment into episodes, where each run through the game counted as one episode. We also introduced a time-step, Δt, so that after the child agent selected an action, the environment would evolve for Δt seconds until a new decision was made. This was to allow for a noticeable change in state from $s_t \rightarrow s_{t+1}$, as well as, allow the game to be played at different speeds. There were many different parameters that had to be tuned and selected to achieve the optimal solution in this game. We implemented the DDQN concept that was mentioned earlier, with a hidden layer of

64 nodes. We used an ε-greedy search strategy for both the parent and child agents. For the child agent, ε started at 1.0 and exponentially decayed until 0.01. For the parent agent, ε also started at 1.0 and exponentially decayed to 0.15. The reason for the parent agent's ε decaying until 0.15 was to allow for the slightly larger probability of exploring the other routes even when many episodes had already been run. In harder game environments, we could obtain a near optimal feasible solution on the non-optimal route. By forcing the lower bound on ε, this ensured that we were able to converge to the optimal route. We used a *tanh* activation function for the child agent and the *ReLU* activation function for the parent agent. This was because for the parent agent, instead of using a multi-layer perceptron network, we used a convolutional neural network whose input was the current game screen. Our memory length for both the parent and the child agents was set to 500,000 to allow for a large number of the previous transitions to be made available for training.

C. *Case Study: two aircraft with four routes*

In this problem, we considered two aircraft where both aircraft have the ability to change routes. Fig. 6 shows the game screen for this problem. With both aircraft able to change routes, this leads to four different route combinations and greatly increases the complexity of the problem. What also makes this problem interesting is that even if the aircraft take the incorrect route, a near-optimal feasible solution can be obtained which makes choosing the route more difficult.

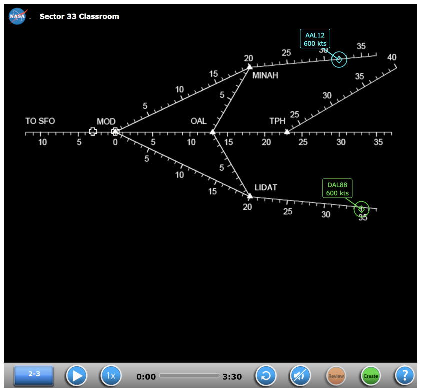

Figure 6. In the case study, both aircraft in this problem have the ability to change routes.

By training the AI agent on around 7000 episodes and choosing a time-step of four seconds, we were able to obtain the optimal solution for this problem. The number of episodes required to achieve the optimal solution was due to the complexity of the problem and the fact that different routes could achieve very similar results. It is also important to note that in this problem, if we chose a different route, the AI agent would achieve the optimal solution specific for that route. Therefore, in a real-world scenario, if there was a last-minute decision to change from the optimal route to a feasible route, the AI agent would be able to maintain safe separation and minimize delay on this new route. Fig. 7 shows the game screen after obtaining the optimal solution to this problem.

D. *Algorithm Performance*

In this section, we analyze the algorithm's performance on the two-aircraft problem. Fig. 8 shows the score per episode during training for the case study.

We can see that throughout training the score tends to oscillate frequently, but is on an increasing trend. This is due to the value of ε being close to 1 and as this value decreases through the episodes, the score increases significantly. We define the score in Fig. 8 to be the cumulative reward at the end of each episode. The score involves the reward of the child agent at each time step, as well as, the termination rewards: -10 for collision, -3 for out of time, and +10 for optimal solution.

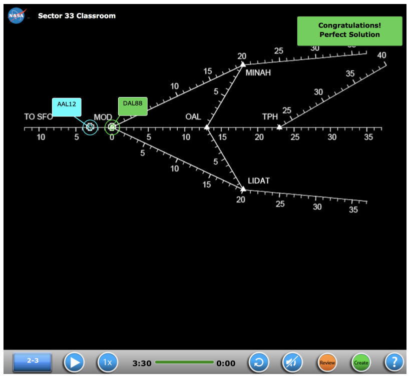

Figure 7. Problem 2-3 of the NASA Sector 33 game environment after obtaining the optimal solution. We can see that both aircraft are on their correct terminal when there is 0:00 left on the timer.

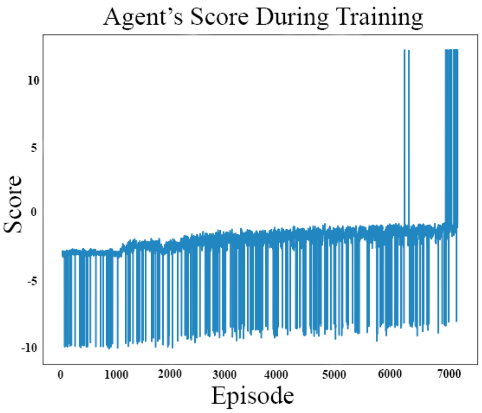

Figure 8. Agent's score throughout the training process for the case study. We can see that as the episode number increases, the score increases as well.

VI. Conclusion

A novel hierarchical deep reinforcement learning algorithm is proposed in this paper to sequence and separate aircraft as a core component in an autonomous air traffic control system. The problem is formulated in a reinforcement learning framework with the actions of changing aircraft route and adjusting aircraft speed. The problem is then solved by using the hierarchical deep agent algorithm. Numerical experiments show that this proposed algorithm has promising performance to help aircraft maintain safe separation, resolve conflict, and arrive at their final metered positions in a case study. The proposed algorithm provides a potential solution framework to enable autonomous sequencing and separation for a fully automated air traffic control system in a structured airspace.

According to our knowledge, the major contribution of this research is that we are the first research group to investigate the feasibility and performance of autonomous aircraft sequencing and separation with deep reinforcement learning framework to enable an automated, safe and efficient airspace. In addition, we propose the novel hierarchical deep reinforcement learning architecture, which is demonstrated capable of solving complex online sequential decision-making problems. The promising results from our numerical experiments encourage us to conduct future work on more advanced air traffic control simulators that can model operational uncertainties.

Acknowledgment

The authors would like to thank Heinz Erzberger for the discussions on autonomous air traffic control and Michelle Eshow for pointing us to the NASA Sector 33 simulation environment.

References

[1] Erzberger, H., 2005. “Automated conflict resolution for air traffic control.”

[2] Erzberger, H., 2007. “Fast-time simulation evaluation of a conflict resolution algorithm under high air traffic demand.”

[3] Erzberger, H., Heere, K., 2010. “Algorithm and operational concept for resolving short-range conflicts.” Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering 224 (2), 225–243.

[4] Amazon, A. P., 2015. “Revising the airspace model for the safe integration of small unmanned aircraft systems.” Amazon Prime Air.

[5] Google, 2015. “Google uas airspace system overview.” URL https://utm.arc.nasa.gov/docs/GoogleUASAirspaceSystemOverview5pager[1].pdf

[6] Kopardekar, P. H., 2015. “Safely enabling civilian unmanned aerial system (uas) operations in low-altitude airspace by unmanned aerial system traffic management (utm).”

[7] Airbus, 2016. “Future of urban mobility.” URL http://www.airbus.com/newsroom/news/en/2016/12/My-Kind-Of-Flyover.html

[8] Gipson, L., 2017. “Nasa embraces urban air mobility, calls for market study.” URL https://www.nasa.gov/aero/nasa-embraces-urban-air-mobility

[9] Holden, J., Goel, N., 2016. “Fast-forwarding to a future of on-demand urban air transportation.” San Francisco, CA.

[10] Uber, 2018. “Uber elevate – the future of urban air transport.” URL https://www.uber.com/info/elevate

[11] OpenAI, 2017. Alphago at the future of go summit, 23-27 may 2017. URL https://deepmind.com/research/alphago/alphago-china/

[12] NASA, 2013. “NASA sector 33 application.” URL https://www.atcsim.nasa.gov/simulator/sim2/sector33.html

[13] Amato, C., Shani, G., 2010. “High-level reinforcement learning in strategy games.” In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. International Foundation for Autonomous Agents and Multiagent Systems, pp. 75–82.

[14] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. “Playing atari with deep reinforcement learning.” arXiv preprint arXiv:1312.5602.

[15] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., 2016. “Mastering the game of go with deep neural networks and tree search.” nature 529 (7587), 484–489.

[16] Watkins, C. J. C. H., 1989. “Learning from delayed rewards. Ph.D. thesis, King’s College, Cambridge.”

[17] Kochenderfer, M. J., Amato, C., Chowdhary, G., How, J. P., Reynolds, H. J. D., Thornton, J. R., Torres-Carrasquillo, P. A., U¨ re, N. K., Vian, J., 2015. “Decision Making Under Uncertainty: Theory and Application, 1st Edition.” The MIT Press.

[18] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al., 2015. “Human-level control through deep reinforcement learning.” Nature 518 (7540), 529.

[19] Van Hasselt, H., Guez, A., Silver, D., 2016. “Deep reinforcement learning with double q-learning.” In: AAAI. Vol. 16. pp. 2094–2100.

[20] Hasselt, H. V., 2010. “Double q-learning.” In: Advances in Neural Information Processing Systems. pp. 2613–2621.