CS229 Project Midpoint
Do A Barrel Roll

Steven Ingram, Tatiana Kuzovleva

December 2014

1 Motivation

We implemented an algorithm to navigate Fox's ARwing space ship from the start of level one to the boss of level one without taking any damage. The problem can be described as navigation of a 3D space with only 2D information. Without actually knowing distance, the craft must consistently avoid crashing into terrain, hitting buildings, trees, and must avoid laser fire from enemy ships. Fox' ship is constrained to go forward at a constant speed and so we can only vary its position by moving up, down, left, and right.

We implemented a reinforcement learning algorithm to train the pilot (Fox McCloud) to navigate a portion of the first level. The goal was to have the ship not crash. We let our algorithm 'play' the game over and over, making use of a simple reward function.

2 Introduction

The problem of navigating 3D space with only 2D information is a common one. The difficulty in our problem lies in the large amount of information we must process during each run of the algorithm and the large number of possible states. We did not classify the objects in our field of view in advance, nor did we access in-game information about the placement of objects in the 3D game space. We only allow the algorithm to process screenshot data.

Figure 1: Sample screenshot

3 Supervised Learning

3.1 Classifying Arrows and Death

Our reinforcement learning algorithm required certain pieces of information to compute the reward function for each state visited. This information indicated whether Starfox had died or had maneuvered too far left, up, or right.

In order to tell when the level had to be restarted, we considered the boost-box. In only two cases does the boost-box disappear. Either if Fox has died or if the boss has been reached.

Furthermore, if arrows appeared on the top of the screen, for example, that meant Fox has flown too far up and the reward for that state was decreased accordingly. Not allowing Starfox to maneuver too far in any direction was a challenging constraint to meet because it would force him to fly near terrain and buildings, but

still avoid them. The problem is much harder than just allowing Starfox to fly high up into the corner of the screen where almost nothing will collide with him.

3.2 Data Collection

We gathered screenshots of both the Boost-box portion and each arrow portion manually. For each classification problem, the boost-box, left arrows, top arrows, right arrows, we gathered over a thousand screenshots of positive and negative examples. Our algorithm then considered only a cropped, relevant portion of each screenshot as input.

3.3 Algorithms

For each set of arrows, the screenshot was converted to greyscale and run through Matlab's edge function. Only 1 in 5 of the 2352 pixels from the cropped region were considered in training and classification. This resulted in each training sample having a dimension of 470. The output for each training sample was manually set to be either 1 or 0, indicating whether one or two arrows were present, or no arrows were present in the cropped region. Finally, stochastic gradient descent was applied to the data to compute theta. The hypothesis we used for $h(\theta)$ was the sigmoid function.

3.4 Results

After computing theta and computing $h(\theta)$ for all the training examples, we determined a threshold for $h(\theta)$ for each case, with which to differentiate if there were arrows or not, or if Starfox had died or not. For determining if there there left arrows or not, the dividing value for $h(\theta)$ is 0.4, for top arrows or not, 0.4, and for right arrows, 0.59. With these values we were able to correctly identify all of the training examples as having arrows or not and if Fox had died or not.

While running our reinforcement algorithm, our hypothesis for arrow identification were accurate over 95 percent of the time. The error in arrow identification occurred consistently when either the exhaust of Starfox's engine was in the region or if enemy laser fire was also in the region. These situations would would cause our hypothesis to incorrectly guess if there was arrow when there actually wasn't. Fortunately, our ability to identify if Starfox died or not was over 99.99 percent accurate. With over 50,000 screenshots, only 2 were misidentified.

4 Emulator Controller

We automated reinforcement learning of Starfox using a sequence of communication steps between three applications: Matlab, Autohotkey, and Project64. Matlab processes the screenshots taken by Project64 of Starfox, computes which next action for Starfox to take, relays that information to Autohotkey, and temporarily pauses. Autohotkey continuously waits for a command from Matlab to issue to Starfox, sends the command using its SendInput function, and signals to Matlab that the Project64 emulator has received the command. Once Matlab is notified that the previous command has been sent by Autohotkey it unpauses and is ready to issue another command. The overall loop of the algorithm is as follows: pause the game, take a screenshot, process screenshot for information, compute action, unpause game, issue action, and repeat.

Matlab and Autohotkey communicated with each other using three files. Two permanent contained the commands from Matlab. The third was created and deleted by Autohotkey to signal Matlab.

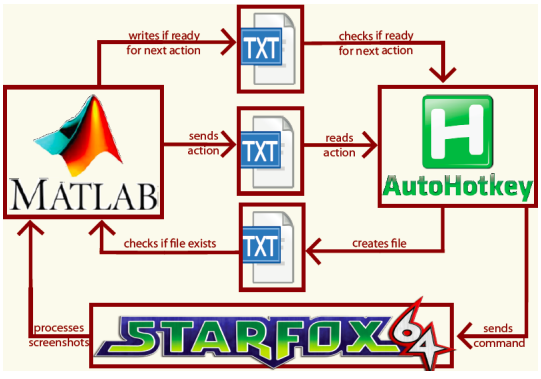

Figure 2: Flow of information

5 Reinforcement Learning

5.1 Data-Set

After data was collected from the Project64 Emulator, our software processed the screenshots. Screenshots were cropped and then converted to gray-scale. We used the Matlab function edge to simplify the input to a 120,000 pixel long logical vector. However, this still left a lot of possible states to explore.

Figure 3: Cropped screenshot

Figure 4: Grayscale image

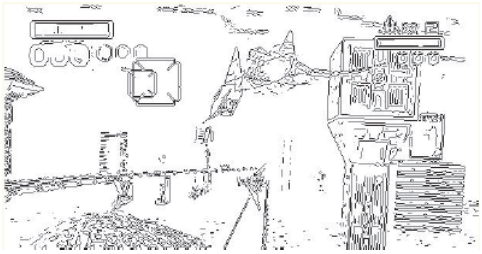

Figure 5: Post edge function

Furthermore, we had to keep track of the health-bar. This was vital for our reward function. We compared the current health bar to previously gathered healthbar images to determine how much health was left. We also registered a hit to our ship by determining if the screen had turned red. We considered only the red content of the image, and if that was above a certain threshold, we registered the ship as hit.

Figure 6: Red screen after collision

5.2 Algorithm

Because of the complexity of the state-space, we implemented a reinforcement learning algorithm to navigate the level. We ran into trouble attempting to express the transition probabilities because of the huge number of states available to our system. Therefore, we chose to implement the SARSA algorithm which allows us to only take into account the action we chose instead of considering all actions available to the ship in each state.

The SARSA algorithm allows us to consider only two states at a time with the following algorithm:

Initialize $Q(s, a)$ as empty Hashmap ;
for *each level of Starfox* **do**
  Fetch and process first screenshot;
  for *each screenshot* **do**
    Choose action based on Q;
    Do action a;
    Take new screenshot;
    Calculate the reward;
    Choose action a' based on Q;
    Update $Q(s, a)$;
    First screenshot = second screenshot;
  end
end

Here, the actual update rule for SARSA is:

$$Q(s,a) := Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$

Unlike most other reinforcement algorithms, SARSA doesn't require us to calculate the transition probabilities for our states, which grealty reduces the time and space needed by our algorithm.

5.3 State-space

The input for our algorithm is the processed screenshot. Because of how complex the game world is, we can not simply boil this down to integer valued attributes. We considered using clustering and supervised learning algorithms to prep the information from our screen capture to allow us to input integer valued parameters. However, we decided to prioritize the reinforcement learning aspect of our project as was suggested in the feedback to our proposal.

In order to speed up access time to our states and action values, we implemented a HashMap in Matlab. Our first implementation used a matrix to store our state values, and we found that searching for states took too long. The Hashmap implementation allows us to play the game in real-time.

5.4 Actions

In the current implementation of the algorithm the possible actions to be taken in any state are movements up, down, left, or right. These movements are implemented through the pressing and holding down of the left, right, up, and down arrow keys.

In the future, we will expand this algorithm to include shooting and doing defencive maneuvers such as the barrel roll which allows the ship to fly through enemy fire unharmed.

5.5 Reward Function

Our reward function penalizes both a loss of health and maneuvers that take the ship too far away from the main path. We assume that no maneuvers increase the health of the ship, and therefore the reward function can have a value of $-a$ any time the health-bar changes at all. This reflects the fact that all changes to health are bad, and should be penalized.

Our reward function also penalizes the ship when it moves away from the main path of the level. Fortunately, the game provides guiding arrows on the screen when you stray too far from the path. Therefore, our reward function takes on a value of $-b$ whenever those arrows appear.

When both these conditions are met, i.e. the health-bar changes and there's an arrow on the screen, the ship has really messed up its navigation, and the reward function is evaluated at $-(a+b)$. Otherwise, the reward function is 0.

$$\begin{cases} -a & StarFox\ hit \\ -b & arrow\ appeared\ on\ screen \\ -(a+b) & both\ health\ and\ arrows \\ 0 & otherwise \end{cases}$$

5.6 Termination

Our algorithm terminated when either we reached the end of the level, or our health dipped to zero and we lost the level.

We measured success of the flight by how much health remains once the level is complete, or how long we stayed alive for, if we did not complete the level.

5.7 Results

We used the following constants without optimization:

$$a = 1$$
$$b = .75$$
$$\alpha = .95$$
$$\gamma = .995$$

We found that Starfox encounters a significant amount of states even when traversing even small portions of the level. The graphs below show in even one simple area of the level, that in every single run of the level, new states were encountered.

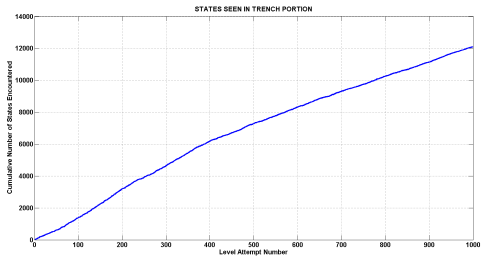

Figure 7: Number of new states encountered in each level

One reason our algorithm encountered a new state at almost every step was due to the fact that communication with Autohotkey combined with Project64 Emulator behavior is not deterministic. Sending the same command in the same state over hundreds of iterations still resulted in hundreds of new states. In our algorithm, despite earlier efforts to reduce the state space as much as possible without resorting to feature classification, all it takes is one pixel out of 120000 to merit creation of new state-action vector.

In the small fraction of steps when a state had been visited before, such as in the initial state or much less often, a state later along the level traversal, our algorithm would look up the reward obtained and add that value to the appropriate state-action pair value. In only a few of the over 30,000 states visited was the ship able to try moving in all four directions.

6 Conclusion

We applied both supervised learning techniques and a reinforcement learning algorithm to attempt navigation of the first level in the videogame Starfox 64. We had high success rate of determing the level of health of Fox, which arrows were on the screen if any, and if Fox had died or completed the level.

Ideally, we will optimize for the various reward constants in our algorithm and determine whether it is possible for a simple reinforcement learning algorithm to navigate the first level of Starfox in its entirety.

7 Future Work

We still hope to find a better way of storing states. As things stand, too much memory is required to store all possible states. Even more importantly, a single pixel difference between two screenshots can cause us to classify the state differently. This requires the algorithm to play Starfox many times before it starts learning. Fox simply does not see states more than once often enough. A different approach to classifying states from screenshots would be worth considering.

We would also like to add more possible actions for Starfox, specifically the famous barrel roll that allows the pilot to avoid incoming laser fire. Other actions like the ability to shoot enemies can be added as well.

The reward function could be expanded to reflect the existence of buffs and save points in the game.

8 Refrences

Kaelbling, Leslie Pack, Micheal L. Littman, and Andrew W. Moore. "Reinforcement Learning: A Survey." Journal of Artificial Intelligence Research 4 (1996): 237-85. Web. 7 oct. 2014.

Liao, Yizheng, Kun Yi, and Zhe Yang. "Reinforcement Learning to Play Mario." (2012): n. pag. Web. 7 Oct. 2014

Eden, Tim, Anthony Knittel, and Raphael Van Uffelen. "Reinforcement Learning." Algorithms. UNSW, n. d. Web. 07 Oct. 2014.

Ng, Andrew. CS 229 Lecture Notes. Part II: Classification and Logistic Regression. 16-19. Web. 12 Oct. 2014.