CS181 / CSCI E-181 Spring 2014 Final Project

David Wihl
davidwihl@gmail.com

Zachary Hendlin
zgh@mit.edu

May 5, 2014

Contents

1 Introduction

To gather sufficient data, we allowed the SampleAgent (provided intially in the code) to play 30 megabytes of ghost training data of games, to get data on feature vectors and associated point values of each.

2 Classification of Ghosts

We sought to classify ghosts as $\{0, 1, 2, 3, 5\}$, where all ghosts in category 5 are dangerous (e.g. induce a reward of -1000 points unless a helpful capsule is consumed first).

We explored two methods for classifying the ghosts on the basis of their features.

First, we explored linear support vector machines using SK-Learn's Stochastic Gradient Descent classifier. For classifying the category 5 ghosts, this approach was accurate 90.62 percent of the time. Our analysis found that while differences in the rewards associated with eating ghosts not from class 5 did differ by class, the most important thing for us to measure our performance on is the correct classification of dangerous (class 5 ghosts).

¡graph here¿

We also used logistic regression classification, and achieved somewhat better results, with correct classification of class 5 ghosts 93.25 percent of the time.

We elected to use the logistic regression classification results to predict which class each ghost is in during runtime.

3 Classification of Capsules and Placebos

We want to determine which of the pills are helpful capsules and which are placebos. To do this, we first plotted the helpful capsules which we collected using the '-d' data collection function.

Here we determined that capsule feature values were very much clustered into three distinct clusters as shown below. ¡show graph¿

This suggested that we could either fit Gaussians to these points as part of a generative approach, or apply something like K-means to find the clusters. Because the data were in three dimensions, we opted to use k-means for clustering, and initiated the model with kmeans++.

Because the three clusters were all positively identified, simply assigning a new value at run-time (for a capsule which may be a 'good' or 'placebo'), we instead used the 'score' attribute to evaluate the distance from the centroid the data point would have been assigned to. We are agnostic to which centroid the capsule would be assigned to but we are very sensitive to the score assigned (e.g. the distance from the centroid assignment), with larger distances being worse.

From our positive training data, we found values of 0 to -118 as the 'good' range of objective function values. Any value < -118 would likely be a placebo capsule.

So at runtime we score each capsule based on its 3×1 feature vector and we consider the 'best' capsule the one with the maximum objective function value when scored.

4 Reinforcement Learning

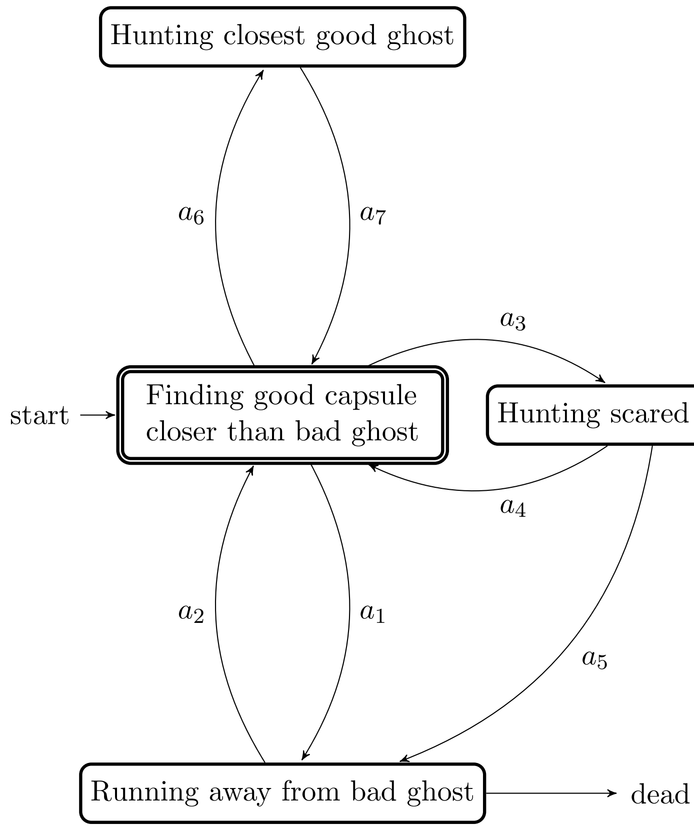
5 Rules Engine

Our next approach was to build a procedural rules engine inspired by a Markov process. In other words, the rules engine did not require any historical data and would simply decide the next action based on the current state of the game.

The other reason for building a rules engine was to better grapple with the game dynamics to see if the model could be significantly simplified. We believe we were successful as the game came down to a straightforward finite state machine.

We iterated through more than 10 different heuristics. Taken over 10 different seeds, our average score over 50 games went from a dismal -7390 to a reasonable 933.

5.1 Finite State Machine



where:

a_1 bad ghost closer than good capsule

a_2 good capsule closer than bad ghost

a_3 bad ghost is scared and within range

a_4 bad ghost is out of range

a_5 bad ghost will no longer be scared soon

a_6 no good capsule in range

a_7 good capsule now in range

The Rules Engine provided the following advantages:

Understanding Game Dynamics We had a much better grasp of the *keys to success* to the game.

Key Factors We could distill key factors that would be difficult if not impossible to discover by exploration, such as the equivalence between clock ticks and Manhattan distance between objects.

Latent Factors One of the key pieces of information was the time remaining before scared bad ghost reverted to normal. We found this by careful examination of the game code. It would have been difficult if not impossible to discover this feature by exploration.

No history With our ghost and capsule classifiers, we could immediately evaluate the best option from the current board.

Flexibility Since only the current state is necessary our rules engine would be flexible enough to handle more dynamics than in the current game. **The entire board could resize, walls could move, capsules could be in motion and the bad ghost could change between time clicks and the Rules Engine would still find a reasonably good action.**

5.2 Parameter Tuning

We defined two configuration settings:

Capsule Threshold As mentioned in the Classification of Capsules, we determined a capsule score. We had to evaluate whether it was worth risking targeting a capsule that might turn out to be a placebo. Interesting, if we played too conservative and tried to reach only certainly good capsules, our average score decreased. It paid to aggressively chase capsules even with a lower certainty. See figure 1.

Time Buffer We knew how many ticks before a scared ghost reverted to a bad ghost, but did not know where precisely the scared ghost would move. So we had to determine the appropriate margin to give the scared bad ghost lest it turn on us. See figure 2.

While both of these parameters could have been found via a regression, it would have required extensive modifications to the provided game execution code to get the appropriate harness in place. We simply opted to manually iterate on these parameters. This involved 21 tuning runs (200 moves, 50 games, 20 different seeds) for over 4M moves used for tuning.

5.3 Scores Across Different Seeds

To ensure the reliability of the Rules Engine, we tested against 20 different seed values $s = \{1..20\}$. While the min and max scores for a given seed could vary significantly, the overall average across multiple seeds was quite consistent once we tuned the parameters.

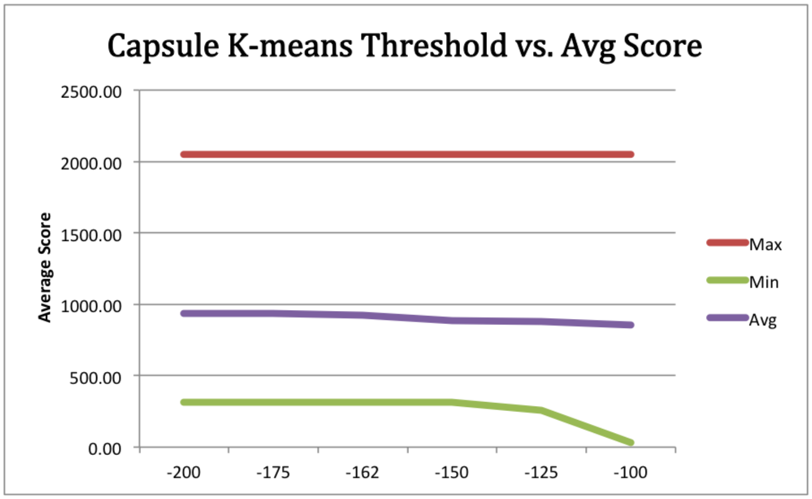


Figure 1: Capsule Threshold Tuning

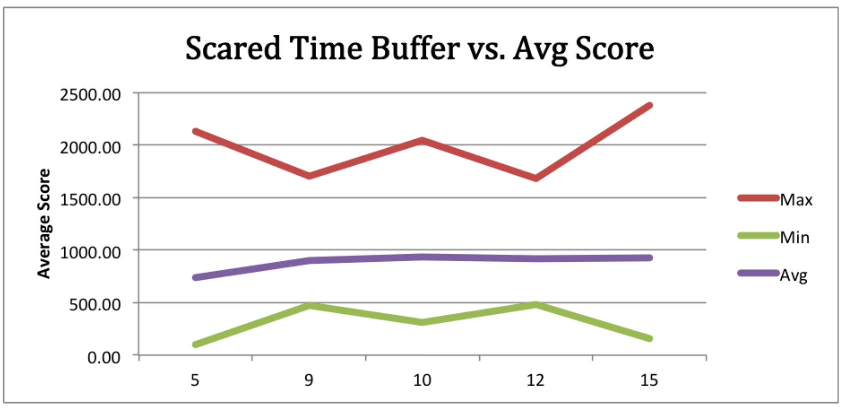


Figure 2: Time Buffer Tuning

Interestingly, the Capsule Threshold improved the minimum score, while the Time Buffer increased the maximum score. See figure 3. Across all seeds we tested, the average score was always positive, giving us confidence in the flexibility of the Rules Engine.

6 Other Methods

Not satisfied with our Rules Engine, we continued to evaluate and experiment with other Machine Learning techniques to find a potentially better solution.

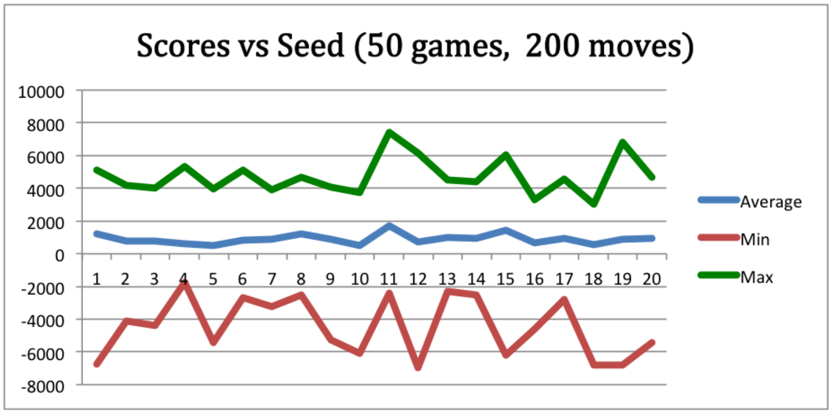


Figure 3: Scores Across 20 Different Seeds, Tuned Parameters

6.1 Expectimax

tree would not be deep to merit this.

6.2 Alpha-Beta Search

Russell & Norvig, pg 166

6.3 Hidden Markov Model

Roland Memisevic http://www.iro.umontreal.ca/ memisevr/code/hmm.py

Needed to find a HMM implementation that respected the dependencies of the game cluster. (For example could not use GHMM www.ghmm.org due to a C library dependency)

6.4 Probabilistic Graphical Model

ref Koller book.

7 Conclusion

A combination of ML and traditional techniques is more powerful than using one set of techniques alone.

References

[1] *Reinforcement Learning*, Sutton & Barto, 1998, ISBN-10: 0-262-19398-1