CS3600 AI Competition

brought to you by

Hack@Tech

a16z

Project 4 (Alternative)

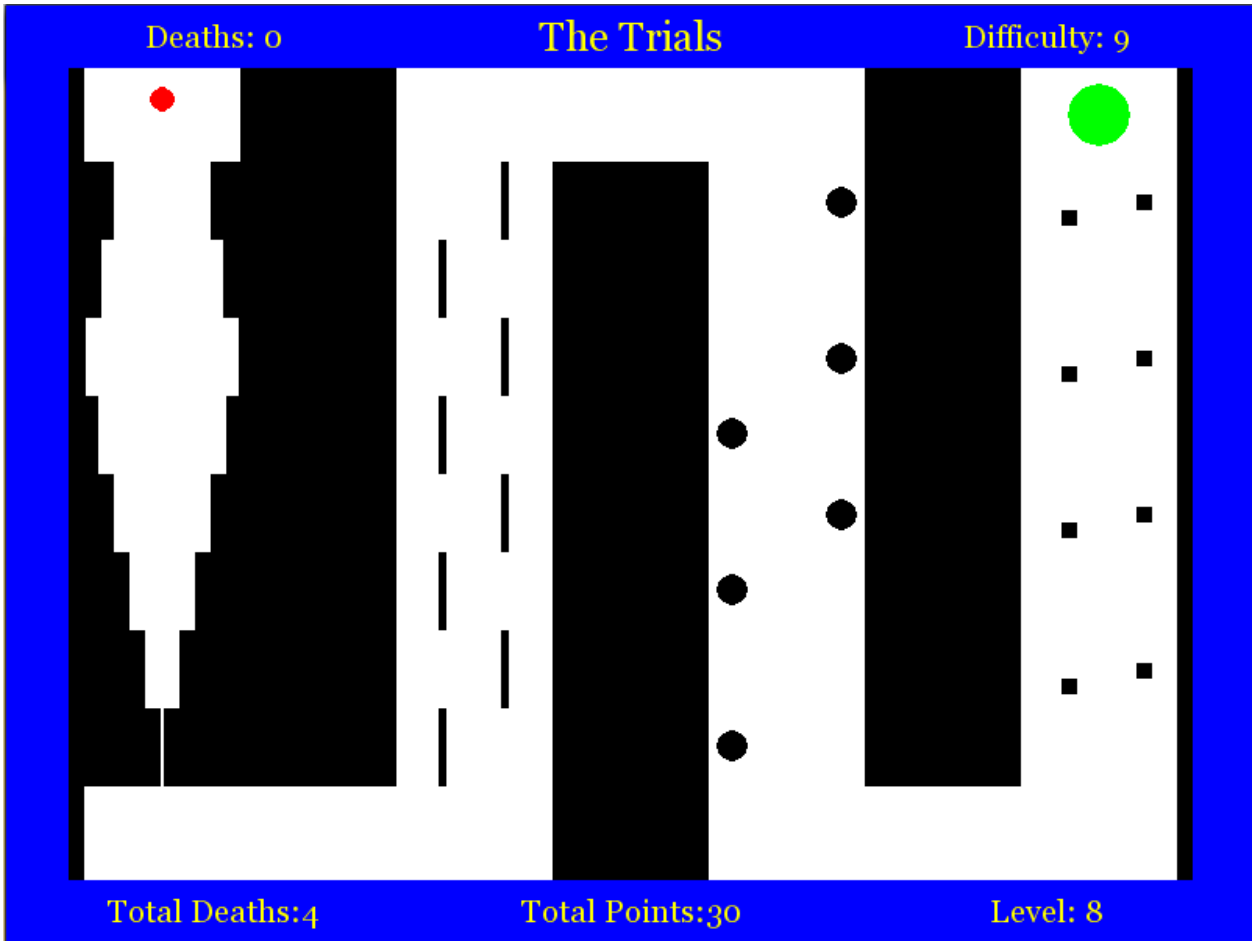
In this competition, you will develop an agent that can play a computer game. The agent must use a machine learning algorithm of your choice to learn to play the game. That is, given a set of possible actions, learn a model that determines which action to perform at which time such that the agent completes the game. Your agents will then be pitted against each other in a chance to win great prizes, and meet a16z recruiters!

This project has two parts. In the first part, you will develop the code for the game-playing agent and test it by playing the game. In the second part, you will provide a write-up discussing the machine learning algorithm and the results.

Notes:

- This project can be completed instead of the regular Project 4.
- By submitting the alternative project, **you are entered into a competition in which the developers of the best performing agents will win prizes provided by a16z**. Prizes are awarded on the last day of class, April 25th from 4-5pm.
- You must declare to the instructor by **April 14th** that you intend to enter the competition.
- Projects must be demoed live to the instructor from 4-5pm on Friday April 25th. Note that this the hour immediately following the last day of class.
- The project will be graded manually based on live demo and code inspection.

- The grade will be in part determined by the performance of one's agent relative to other students' agents. Grades will not be determined by a strict ranking, but will be determined by the extent to which an agent outperforms or underperforms the mean performance.
- The competition results and project grade will be assessed independently of each other.



This project will be done in Python. Your submission should consist of your code and your writeup. For the code, it can be implemented in mouse_game.py and other files that you create. There is no autograder. For the write-up, publish it as a doc, rtf, or pdf document and include it with your submission as a separate file.

Files you will edit

- mouse_game_with_agent.py --- This file contains the agent control interface. Implement your agent here.
- You will likely want to implement your machine learning algorithm in a separate file—which is run offline—and feed the results into the game agent.

Files you will not edit

- All other files that come with the game distribution.

Your objective is to develop a machine learning algorithm that learns to play the game without human help. That is, the agent that plays the game will make a decision at every move based on a model produced from play trace data. You may implement any machine learning algorithm you believe will work best. There is no data set provided. **You must generate your own data set** by manually playing the game and recording play trace data. We recommend doing this on a good computer, and NOT a VM, as the saving may slow down the machine or crash the game. Also, make sure you have ample space on your machine.

To get started, go into the Source directory. mouse_game.py is used to play the game and generate Frame and Label files within the data folder. The Frame files are binary files that can be loaded into python programs as Numpy arrays. The Label files are simply tuples for the vector direction that the ball should move in that frame.

There is no data set given. You must generate your own data from the game and use it to train your machine learning algorithm. You can train your agent on any of the maps provided in the distribution.

In the Source folder is mouse_game_with_agent.py, which is the file that you will edit to implement your agent and machine learning algorithm. There is a method called *agent* in which one should return a vector as a tuple for the ball to move, given a numpy array of the current screen pixels. Modify the *agent* method.

Note that you may implement any machine learning algorithm. The agent method is to perform the **result** of your model. You must implement, in python, the learning algorithm elsewhere and simply include the model created, and code to use the model to output a result, within the agent method. It is, in fact, recommended to do this in another file. There is a lot of liberty in the format of your code for this other file. However, you **must** implement the algorithms yourself.

Support for this project is minimal. You will be expected to figure out how to run the game and how to control the agent on your own. The course TAs will have had minimal familiarity with the project before hand, but may be able to provide suggestions and help for Decision Trees and Neural Network implementations if you choose to use one of these techniques. Please post on Piazza for game infrastructure questions.

Neither the TAs nor the instructor know what algorithm will work best. So this really will be a novel solution on your part! Try to experiment and be creative. There are multiple great prizes on the line so good luck and have fun!

Write-up:

The write-up is 60% of the project grade. Your write-up should cover three topics:

1. You should justify your decision for your machine learning approach. Explain why you chose the particular technique, why you felt it was the most appropriate choice, and why other alternatives were deemed to be less likely to succeed.
2. You should describe your machine learning implementation. Give the specific algorithm, plus any modifications you made to the algorithm to specialize it to the game playing agent. Provide details on where in the code to find particular functionality of the algorithm.
3. Based on your observations of the performance of the agent, explain whether you think the algorithm was a good choice for the game playing agent or not. Explain why you think it was a good choice or a bad choice. Provide details of error rates in training and testing. Provide numerical details to back up claims of success (or failure) of the agent when playing the game.

Minimum length of the write-up is one page (12 point Times New Roman font, with 1-inch margins). Provide as much detail as necessary to understand the implementation and provide analysis of your agent's performance. Be as specific as possible.

Agent Performance:

Your agent implementation is worth 40% of the project score.

Evaluation of your code will happen in two parts. First, you will arrange to demonstrate your game playing agent to the instructor. The average performance of your agent will be used for the competition. The agent will be tested on **map1** plus two other maps that are not provided in the distribution but are similar in nature to some of the other maps provided.

Your grade will be determined by the average performance of your agent relative to the performance of other students' agent performances. Specifically, the instructor will cluster agent performances and assign grades to each cluster. The top performance cluster (or outlier) will receive full credit. The next cluster will receive some penalty, and so on. The exact scoring metric cannot be determined until all projects have been analyzed.

Second, the instructor will manually inspect your code to verify the machine learning algorithm implementation.

Tips:

- The mouse game generates data every time the screen is rendered (many times per second). This produces greater than 1 GB of data per map played. You may want to edit the game code to reduce the frequency that data is recorded.

- Data contains the color of all pixels on the screen. You will need to implement a means of extracting features (e.g., distance to nearest wall) from the screen. You can assume perfect "sensor" accuracy. Alternatively, you may want to modify the game so that features are computed at runtime and recorded instead of complete pixel dumps. Reducing the complexity of the environment to a number of easily computable features is likely to be key to a practical solution.
- It may be possible to implement a reactive or A* agent that performs reasonably well on the game. This is prohibited. Valid submissions must learn which actions to take at each time step.
- The purpose of withholding two maps is to help verify that your machine learning agent is generalizing. You algorithm must be able to do a reasonable job on maps that it has never seen. Thus, if you over-fit on your training data, you will likely do well on map1, but poorly on the other maps. Generalization (possibly through dimensionality reduction) is thus a key goal.