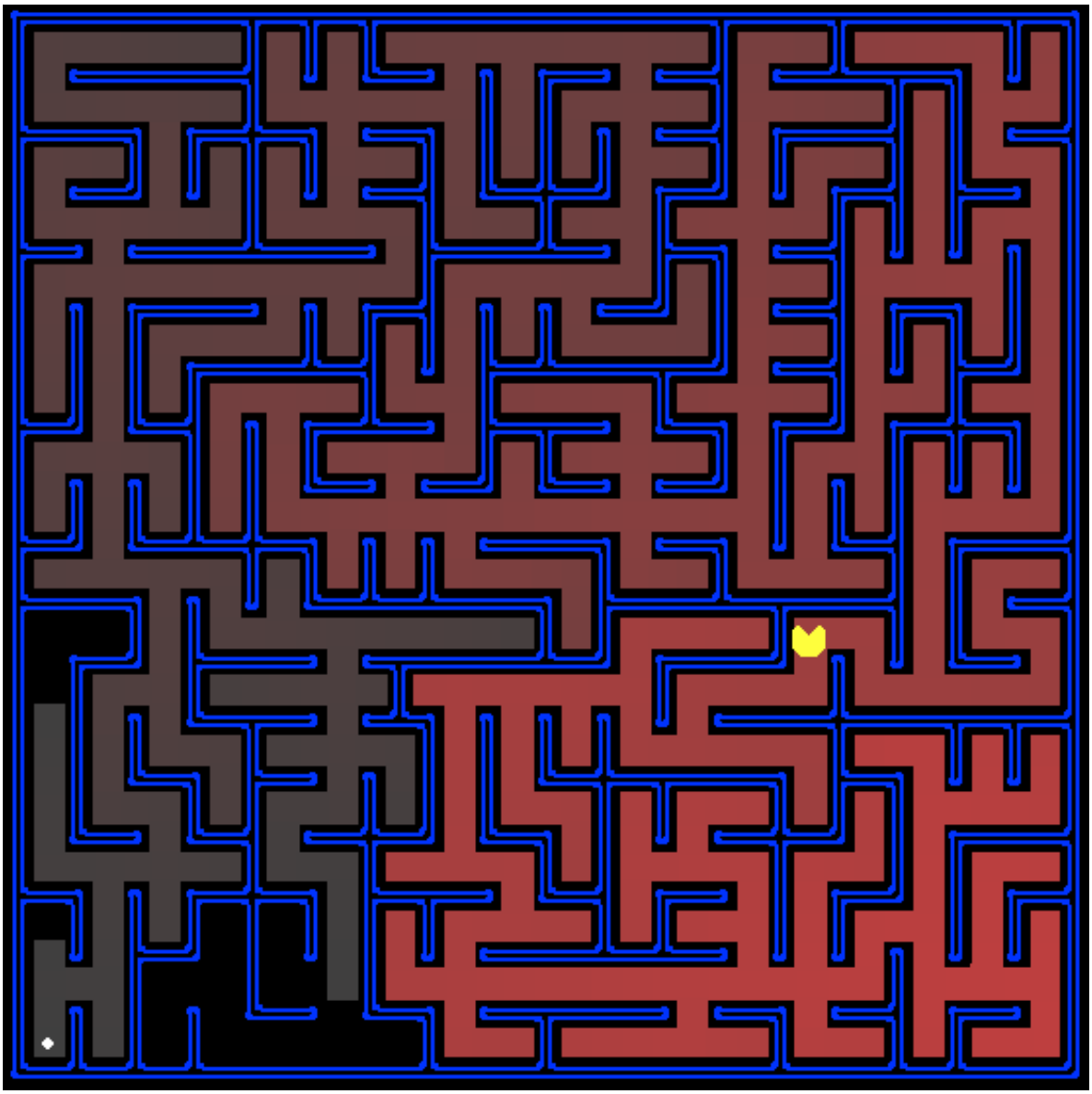
Homework 2 of CS 165A (Fall 2020)

- Assigned on Feb 4, 2021 (Thursday) Due at 11:59 pm on March 9, 2021 (Tuesday)
- Be sure to read "Policy on Academic Integrity" on the course syllabus
- Any updates or correction will be posted on the course Announcements page and piazza, so check there occasionally
- You must do your own work independently. We will use software to automatically detect any plagiarisms.
- TA in charge of this homework: Kaiqi Zhang (kzhang70@ucsb.edu)

Search and Games



Introduction

In This project, you will design agents for the classic version of Pacman, including ghosts. Along the way, you will implement both reflex agents and minimax search and try your hand at evaluation function design.

Before you start programming, consider how to formulate the PAC-man into a artifical intelligence problem, i.e., what are the actions you are supposed to make? How many states are there in total?

How are the actions leading to transitioning of the states?

We suggest you using python3 (python3.6, python3.7) for the project. If you want to use C++ for programming, please read (https://docs.python.org/3/extending/extending.html) on how to write an Python module using C++. Please talk with me so that I can set up autograder for you.

The code for this project consists of several Python files, some of which you will need to read and understand in order to complete the assignment, and some of which you can ignore. You can download and unzip all the code and supporting files from [search_and_games.zip]

Files you will edit.

`multiAgents.py` Where your search-based agents will reside. Please modify the `MultiPacmanAgent` class.

Files you might want to look at

`pacman.py` The main file that runs Pacman games. This file describes a Pacman GameState type, which you use in this project.

`game.py` The logic behind how the Pacman world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.

`util.py` . Useful data structures for implementing search algorithms.

Files you will not edit

`agentTestClasses.py` Specific autograding test classes `graphicsDisplay.py` Graphics for Pacman `graphicsUtils.py`Support for Pacman graphics `textDisplay.py`ASCII graphics for Pacman `ghostAgents.py`Agents to control ghosts `keyboardAgents.py`Keyboard interfaces to control Pacman `layout.py` Code for reading layout files and storing their contents `autograder.py` Project autograder `testParser.py`Parses autograder test and solution files `testClasses.py`General autograding test classes `test_cases/`Directory containing the test cases for each question

Files to Edit and Submit: You will fill in portions of `multiAgents.py,` during the assignment. You should submit this file with your code and comments. Please *do not* change the other files in this distribution or submit any of our original files other than this file.

Evaluation: Your code will be autograded for technical correctness. Please *do not* change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation – not the autograder's judgements – will be the final judge of your score. If necessary, we will review and grade assignments individually to ensure that you receive due credit for your work.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; *please* don't let us down. If you do, we will pursue the strongest consequences available to us.

Getting Help: You are not alone! If you find yourself stuck on something, contact the course staff for help. Office hours, recitation, and Piazza are there for your support; please use them. If you can't make our office hours, let us know and we will schedule more. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

This assignment is based on the Pacman AI projects developed at UC Berkeley, http://ai.berkeley.edu.

Welcome to Pacman

After downloading the code ([search_and_games.zip]), unzipping it, and changing to the directory, you should be able to play a game of Pacman by typing the following at the command line:

python `pacman.py`

Pacman lives in a shiny blue world of twisting corridors and tasty round treats. Navigating this world efficiently will be Pacman's first step in mastering his domain.

The simplest agent in `searchAgents.py` is called the `GoWestAgent`, which always goes West (a trivial reflex agent). This agent can occasionally win: python `pacman.py` –layout testMaze –pacman GoWestAgent

But, things get ugly for this agent when turning is required: python `pacman.py` –layout tinyMaze –pacman GoWestAgent

If Pacman gets stuck, you can exit the game by typing CTRL-c into your terminal. Soon, your agent will solve not only `tinyMaze`, but any maze you want. Note that `pacman.py` supports a number of options that can each be expressed in a long way (e.g., `--layout`) or a short way (e.g., `-l`). You can see the list of all options and their default values via: python `pacmen.py -h`

Note: if you get error messages regarding Tkinter, see this page

In this assignment, You need to implement a search agent for pacman. There can be more than one pacman in the maze. In this case pacmans need to both collaborate and compete against each other: their scores are counted separately, but as long as one pacman is dead, the game ends. You're free to use any search algorithm (minimax, expectimax, A*, etc.), and design your heuristic if needed. The only requirement is that the search depth must not exceed the parameter given to you. A search depth of 1 means every agent (pacman and ghost) in this maze move one step, and a search depth of 2 means every agent (pacman and ghost) in this maze move alteratively and each agent move two steps, etc. The pacmans will always move first. If your search depth exceed that requirement, you'll submission will be void and you'll get 0 credit for Gameplay performance part.

You can run `python pacman.py -p MultiPacmanAgent -a depth=3 -g DirectionalGhost` locally to see how your search agent works.

Grading

Grade Breakdown:

- 20% Complete Report
- 10% includes Makefile and/or executable that matches specification.
- 10% accepts command line arguments; produces required outputs; no segfaults, exceptions, or logic bugs during gameplay
- 60% Gameplay performance

Plagiarism Warning: We are going to use software to check plagiarism. You are expected to finish the project by yourself without using any code from others.

Your pacman will be evaluated in a medium mazes for 10 times. It will be the only pacman in this maze, together with a few ghosts rushing to you with probability 0.5 and random walking with probability 0.5. As long as you win at least once (eating all the foods), you get 60% of the credits for gameplay performance part. If you get more than 1500 scores on average, every 10 score worth 1% of the credit in this part. The extra credit will be based upon the leaderboard and tournament.

Leader board

Your pacman will be evaluated in a set of different mazes for 10 times. and the total score will be used in the leaderboard. The runtime will be recorded as well. If there is a tie in scores, the one with the least runtime will take the lead. Top 3 in the leaderboard will get extra credit.

Tournaments

Participation in the Tournaments is optional but can gain additional credit (and fame!). Due to the limitation of gradescope, tournament results can only be hold offline. If you want to participant, please include a text file named `TOURNAMENT`. The tournament will be held weekly after getting at least 3 valid submissions that would like to participant, and the result will be announed on Piazza.

Submission

Complete as specified in the project instructions. Then upload `search.py` to Gradescope.

We will evaluate your code with **a variety of** test cases. Those test cases on the gradescape are not restricted to those that are provided to you, please pay attention to corner cases. The **hard-coded** solutions will not work on gradescope and we will use software to automatically detect any plagiarisms. The autograder on Gradescope might take a while but don't worry: **so long as you submit before the due date, it's not late**.