

# Koç University

## COMP341

### Introduction to Artificial Intelligence

### Assignment 2

Instructor: Barış Akgün

Due Date: December 1 2024, 23:59

Submission Through: KUHUB Learn

**Make sure you read and understand every part of this document**

## Introduction

This programming assignment will test your knowledge and your implementation abilities of what you have learned in the adversarial search part of the class. You are asked to complete a coding part and answer a few questions about how it runs. The coding part of the homework will follow the Berkeley CS188 Spring 2024 Pacman Project 2: Multi-Agent Search at <https://inst.eecs.berkeley.edu/~cs188/sp24/projects/proj2/>.

This homework must be completed individually. Discussion about algorithms, algorithm properties, code structure, and Python is allowed but group work is not. Coming up with the same approach and talking about the ways of implementation leads to very similar code which is treated as plagiarism! Furthermore, do not discuss the answers directly as it will lead to similar sentences which is treated as plagiarism. Any academic dishonesty, will not be tolerated. **By submitting your assignment, you agree to abide by the Koç University codes of conduct.**

**Important:** Make sure you go over all the submission instructions at the end of this document before submission. Once you upload your submission, download it to make sure it is not corrupted and it has your latest report/code. You are only going to be graded by your KUHUB Learn submission.

**Warning:** The solution code for the homeworks can be found online. We are going to compare your submission with these sources. We are also going to compare your code to latest and previous submissions of Koç students. If your code's similarity level is above a certain threshold, your code will be scrutinized. If there is strong suspicion of plagiarism, we will take action based on university policies.

## Programming

You are going to do the 5 programming questions about adversarial search given in the website. You are only required to change *multiAgents.py*. If you have any issues with other parts of the code let your instructor or TA know ASAP, even if you manage to solve your problem. Use the data structures in *util.py* for the autograder to work properly. If you think you have the right answer but the autograder is not giving you any points, try to run it on individual questions (look at P0 for details on how to use the *autograder.py*).

## Hints

There are hints for the programming part, especially about what features to use in your evaluation functions, in the project description website. Read those carefully, as they generally provide good ideas. We have also discussed example features during the class. The first things to try are the distance to ghosts, distance to food, distance to capsules, scared timer etc. More information is given below. Note that your evaluation function for question 1 is a good starting point for question 5. As noted in the website, do not waste too much time on both of these questions and only get back to them if you have time left.

## Evaluation Functions

The `evaluationFunction` method of the `ReflexAgent` class already shows you how to get important information from the `GameState` class. As an addition, you can use the `getCapsules` method to get the

locations of the remaining capsules as a list. These capsules let the pacman eat the ghosts for significant additional points!

You can call the `asList` method of the `Food` class to get the locations of the foods as a list. You can use the built-in `len` function to get the remaining number of foods or capsules. The `scaredTimes` gives you the remaining time for pacman to eat the ghosts. A 0 value means that the ghosts are not scared so watch out!

You are potentially going to use the distances to the food as a feature (e.g. the distance to the closest food). Your agent, pacman, might get right next to the food and not eat it! This is due to the fact that your evaluation function might return a lower value after eating the food and the next distance to the closest food or the average distance to the foods becomes higher. Make sure your evaluation function takes this into account. The same thing might happen with eating the ghosts, pacman might chase the ghost but never actually eat it.

In the previous homework, HW1, you had the option to use the `mazeDistance`, which gives you the cost to go between 2 locations in the maze taking the walls into account, i.e., the true graph cost. This results in better cost estimates at the expense of more computation. This resulted in lower number of node expansions but potentially made the algorithms run slower than if you used the `manhattanDistance`. Feel free to use either, however, make sure that your code does not take too long!

Your pacman can get to a state of thrashing, which means that it either stops or keeps moving between two states, until a ghost comes nearby. In your `ReflexAgent`, you might think of penalizing the `Stop` action. This is not the best idea and there are other things you can do. If you happen to have this problem, try to tune your feature weights. Finally, it is okay if you do not have the best agent behavior as we are not sending a rover to Mars, yet.

### Programming for Multiple Ghosts

Remember that in the class, we talked about the multiplayer case, where all the agents are trying to maximize their utility. Think of all the ghosts as trying to minimize your evaluation function, i.e., all the ghosts are *MIN* agents. This means that you need to call the *min-value* function for all of the ghosts. You can do it by passing an agent index to your value function. Note that pacman will always be agent 0. Look at the comments in the `multiAgents.py` file to see which functions take an agent index (look for `agentIndex`) as a variable. You are free to define your own functions, either externally or within the classes, but make sure the entry points are in the existing class methods (e.g. the `getAction` methods).

### Keeping Track of Depth

In this assignment, 1-level of depth is interpreted as including all the agents' actions. In the 2-player version, this means 1-level of depth is completed after you check the actions of both the *MAX* agent and the *MIN* agent. If you have more than 1 ghost, the depth increases after you look at the actions of pacman and all the ghosts.

In the class, we have discussed why we are not able to search as deep as the end of the game. The assignment requires you to implement a depth limit. In a recursive algorithm, it might not be obvious how to implement this. There are multiple ways to do this. You can pass down a depth variable, along with your state to your functions.

### Alpha-Beta Pruning

The  $\alpha$  and the  $\beta$  values are not for the entire search tree but for a certain node. This means that you should not keep the same variables for the entire search.

### Pacman Not Eating the Last Food

Due to the setup of the environment, you might not be able to force the pacman to eat the last food for the programming question 5. This is because the action sequences *Stop* and *Move*, where the latter is moving in any legal direction, and *Move* and *Stop* will potentially have the same value at the end of the game depending on the location of the ghosts. Since the *Stop* always comes before the *Move* actions when getting the legal actions, your agent might always chose it first. Keep this in mind if your pacman stops right next to a food at the end of a game. A way to solve this is to force pacman to chose the *Move* action in case two actions have the same value.

# Large-Language Model Usage Report

As mentioned in the syllabus, you are allowed to use Large-Language Models (LLMs) such as ChatGPT, as long as you cite your usage. Towards this end, you are asked to prepare a short usage report if you used these. This report should include for which questions and how you used LLMs. The how part can be achieved by adding a few example prompts and their outputs. I am giving a few examples below:

“I used LLMs to get help on Python. I was getting `unhashable type: 'list'` error while trying to create an explored set. Here is how I used the LLM:

ME: I am getting an unhashable type: list error while I am trying to populate a list. How can I solve this?

LLM: The “unhashable type: list” error usually occurs when you try to use a list as a key in a dictionary or as an element in a set. This is because lists are mutable (changeable) and therefore unhashable, meaning they can’t be used as dictionary keys or stored directly in a set. To fix this error, check if you’re accidentally trying to add a list as an element in a set or as a key in a dictionary. Instead of using a list, you can use a tuple (which is immutable) if you need a similar structure but also need it to be hashable.

As a result, I have changed my state representation from a list to a tuple and fixed my issue.”

You can also describe it as a narrative without all the interaction:

“I used LLMs to get an implementation of the depth first algorithm. My prompt was “....” followed by the incomplete code I needed to complete. The response was not compatible with the code base so I additionally sent this “...”. Finally, I modified line 7 so that the dictionary has the correct values.”

It does not have to be code, you can use it for math as well:

“I forgot how the quadratic formula and this is how I used ChatGPT

ME: Can you explain how to solve a quadratic equation using the quadratic formula, with an example?

LLM: To solve a quadratic equation of the form  $ax^2 + bx + c = 0$  using the quadratic formula, you apply the formula  $x = (-b \pm \sqrt{b^2 - 4ac}) / (2a)$ . For example, for the equation  $x^2 - 3x - 4 = 0$ , we have  $x_1 = \dots$

Prepare and send a report with the aforementioned information. If the report looks like it is going to be too long, tell us where you used it and only provide a few examples. The format of the report is not too critical but it needs to clearly show how you used LLMs.

## Submission

You are going to submit a compressed archive through the KUHub Learn site. This archive should only contain *multiAgents.py* and optionally *report.pdf* if you used LLMs. Make sure **you put your name and ID inside the report** as well. Other files will be deleted and/or overwritten.

## Submission Instructions

- You are going to submit a **single** compressed archive through the KUHub Learn site. The file can have *zip*, *rar*, *tar*, *tar.gz* or *7z* format. If you submit files individually they may get lost.
- You are fine as long as the compressed archive has the required files within 4 folder levels.
- Code that does not run (e.g. due to syntax errors), that does not terminate (e.g. due to infinite loops) or that blows up memory will not get any points.
- **Important:** Download your submission to make sure it is not corrupted and it has your latest report/code. You are only going to be graded by your KUHub Learn submission.

Best of luck and happy coding!