AI Programming (IT-3105): Basic Expectations of the Structure and Behavior of Project Code

1 Introduction

Computer programming can be a healthy mixture of art and science. Some computer-science disciplines attempt to enforce a very rigid (*scientific*) style upon programmers, as this often makes them more compatible with their colleagues in industry, where *rogue* coders typically get little affection. Artificial Intelligence (AI) has a long history of bucking those trends, of letting programmers listen to their inner creative voices and of coding in a more bottom-up than top-down manner. The renowned programmer, tech blogger and author of several AI programming books, Paul Graham, once wrote that *AI Programmers are often undisciplined, like airline pilots who refuse to file a flight plan before taking off.*

The aim of this course is not to strangle your creative urges with a rule book full of mandatory coding principles, nor to launch a movement in free-form, artistic hacking. There is a middle ground, and the level of discipline therein will be required of you.

That middle ground involves four main characteristics that your system must exhibit: generality, modularity, flexibility and transparency. Code written with these factors in mind should be easy to maintain, extend and demonstrate. Complex definitions of these terms are unecessary; simple ones should suffice:

- Generality - the system should be applicable to many different problems or domains.
- Modularity - the data structures and procedures of the code should be encapsulated in modules (e.g. objects, methods, etc.) such that most complex interactions occur within these modules, not between them; this, in turn, permits the seamless addition of new modules during system extensions.
- Flexibility - running the system on different problem scenarios should require the bare minimum of adjustment, and only at the interface level.
- Transparency - the overall behavior of the system in operation should be easy for an observer to discern. Specifically, problem-solving progress should be visualized in various ways so that an observer can quickly assess whether or not the system properly solves the problem at hand.

The relationships between these four aspects are elaborated in the discussion that follows.

2 The Critical Divide

Most of the AI systems that you will design and build for this class will involve an explicit *search* process, in a *state space*, where your code must handle both the search algorithm and the representation of state space. The former must be general enough to handle many instantiations of the latter. This is the basic definition of a *general-purpose AI system*.

To achieve this generality, your system must display a very basic form of modularity: the logic of the problem domain (and thus the contents of and relationships between problem states) must be **very cleanly** separated from the code of the search process. Thus, your system must contain a separate unit (preferably a class) specialized for a given domain (such as the game of Othello). Let's call this class the *Simulated World* or *SimWorld*. It should contain code that:

1. *Understands* game states and the operators that convert one game state to another.
2. Produces initial game states.
3. Generates child states from parent states using the legal operators of the domain.
4. Recognizes final (winning, losing and neutral) states.

This means that your AI code should have **no** explicit references to a particular problem domain. Rather, it should interact with a SimWorld object that responds to messages such as *produce-initial-state*, *generate-all-child-states-of-the-current-state*, and *is-the-current-state-a-final-state*. Then, for each new problem domain, you will create a new SimWorld that handles these same messages in a domain-specific manner. For example, if you build a general-purpose MiniMax algorithm, then you will build separate SimWorlds for the different games played by Minimax: chess, checkers, Othello, etc.

To make this separation explicit, write each SimWorld in a separate file, and, of course, write the AI search process in another distinct file.

Unless otherwise stated in the project description, your code **must** have a very clean and explicit separation between the SimWorld and the AI (Search) code. **Failure to follow this simple design principle can incur a very significant point loss for a project, possibly as high as half the total points for that project.** This is one of the first things that we look for when examining your code during a demonstration session.

3 The System Interface

Your system must be designed for a very basic form of flexibility: many runs (in the same problem domain but with varied parameters settings for the domain and/or search algorithm) can be done quickly, without wasting time hunting through the source code to make simple modifications (to the aforementioned parameters). Therefore, many pivotal parameters of the system need to be determined in an *interface*, where this term is used very loosely.

You are welcome to build a nice graphical user interface (GUI) for entering these parameters. However, a GUI is normally NOT a requirement, unless specifically stated in the project description. An easier alternative is a configuration file (of any format you choose) housing values for the pivotal parameters. Another option is to simply code up a main routine that takes all of the pivotal parameters as arguments. Any of these choices are acceptable, and none is worth any more or less points than another.

However, points will be lost if defining a new scenario requires you to make changes in many source files (or many distant places in the same source file). In short, insure that all pivotal parameters can be set in ONE PLACE.

The *pivotal parameters* will vary among the different projects; they will be explicitly stated in the project description.

4 Visualization

This term is also defined very loosely for the purposes of this course. To clearly show that your system behaves properly, a simple command-line graphic may suffice. However, in most cases, a more elaborate display will be necessary for presenting problem states such that an observer can quickly and easily understand them. The old saying that *a picture is worth a thousand words* is hardly an overstatement.

Furthermore, the observer is often you, the programmer, as you go about your normal coding and debugging. A nice, informative graphic display can reduce your own development time considerably, by making vital run-time information *pop out*. Thus, the investment in a nice display is normally well worth the effort, particularly when done early in the coding process.

During the demonstration session, the reviewer of your system **must** be able to easily interpret its problem-solving progress. In many cases, this simply means being able to see a game board and its changing states, or the gradual change in key dependent variables such as loss (for deep learning) and fitness (for evolutionary algorithms). The former calls for a graphic display of the game board, while the latter motivates a simple two-dimensional plot of loss/fitness as a function of the problem-solving step (e.g. epoch or generation, respectively).

When a project requires a particular form of visualization, that will normally be explicitly written in the project description. However, many situations are quite obvious. For example, any AI system for solving puzzles or playing games **must** be capable of displaying game states in an easily (and quickly) digestible manner. It must also include a flag for turning the game-state viewer on/off, since explicit visualization during the entire search process (involving hundreds or thousands of games) is obviously not desireable. During a typical demonstration session, visualization will be turned off while the system learns a good strategy. Then, you will turn visualization on for a round or two of game play to show that the system plays legally, and hopefully intelligently.

It is to your great advantage in this course to become familiar with a set of graphic tools, such as Python's matplotlib, so that you can easily tackle any of the visualization requirements of a project.

Failure to implement a minimal display (on the command line or in a graphics window) that allows the demonstration-day reviewer to easily interpret your system's progress and problem-solving ability will result in a significant point loss, since our assessment of your work requires straightforward verification of its behavior.

5 Independent Work

This is a course in programming, not downloading. Most of the key components of your systems must be implemented from scratch, not from GitHub. You are free to *learn things* by reading online or by discussing a project with other students, but you should write the final code yourself.

During demonstrations, you may be asked to explain your code. If you are working in a group, then you need to be able to explain ALL of the code, not just the part that you wrote. Any significant weakness in the ability to explain code can result in a very severe penalty, possibly as much as 100% of the points for that project.

Any indications that code has been copied between students or groups can also incur a serious penalty, for both the copier and the code provider.

If you find or develop a *useful tool* for a project and want to share it with the class, then ask the course instructor before doing so. This type of sharing will be restricted to items outside of the *AI core* of the project. For example, some advanced graphing code that uses matplotlib might be acceptable, while a version of Minimax or of Temporal Difference Reinforcement Learning would almost certainly not be.

6 Conclusion

Nothing is this document is new, and all or most of it is probably familiar to all students at your level of study. None of the terms above have strict formal definitions in computer science, but all constitute very fundamental principles that any serious computer scientist takes for granted. Still, the code that one finds online often fails miserably to satisfy these requirements, particularly those of generality and modularity. In their defense, people who post code online are often doing so to help fellow programmers navigate the intricasies of a particular algorithm or software tool. In those cases, their code serves to illustrate key concepts of the algorithm or tool. Writing code *by the book* would only complicate issues, drowning those key concepts in a jungle of classes and methods.

The important thing is that you understand the difference between these two styles of coding: one for explaining a specific concept and the other for building a general-purpose system. In this course, your focus must be squarely on the latter. Major failures to do so will be penalized by point loss. These *rules* are quite easy to follow for anyone with an intermediate computer.science background....and I am confident that there are still many ways to *go rogue* while remaining within these boundaries.