Keeping track of things

There are four basic parts to the course: sensing, movement, decision, and learning. Also hidden in there is the notion of NPIC character development. Of course there is also putting these items together in some sort of Sense-Think-Act cycle where the following, at least a partial list, of soft constraints are placed on the adequacy of an implementation:

The NPIC should

* be intelligent, yet purposely flawed
* be goal driven: decide and then plan
* perform within the constraints
* have knowledge: knows about the world and uses the knowledge in decision making
* be consistent: the agent keeps a believable consistent character
* have no unintended weaknesses
* be responsive: responds quickly to world changes
* execute efficiently: low CPU and memory usage
* be configurable by game designers or players
* be practical: easy to develop, change and deploy
* not keep the game from shipping

With at least some of these ideas in mind, you will need to take items from your tool-kit to provide specific solutions to specific problems. Now this assumes that you have a tool-kit. This is about forming that tool kit.

Design-patterns are used in designing OO software. But the notion is more general than that. So let’s expand on the OO notion of a pattern and maybe add a bit of algorithmic flavor as well.

A design pattern is intended to describe a problem that occurs over and over again in a given environment and the core of the solution to that problem. The descriptions should be sufficiently broad and abstract so that the specific implementations of a solution for a specific problem can vary.

Remembering that an algorithm is a set of well-defined instructions for carrying out a particular task, the solution description need not be an algorithm, but can be algorithm-like. It should be a description of the general steps to be taken to solve the general problem. For example, in the pattern one might say if you need things in order, use a sort procedure with a specific ordering rule. In applying the pattern to a specific case you might use the heap sort algorithm. Following that decision you would implement the algorithm. Note that you might also specify the use of heuristics in a pattern, where a heuristic is a technique that helps you look for an answer.

In order to get our tool kit or notebook started, we will need to add a few things.

The first thing to add will be to add sections that gather different items together in terms of the four main parts of the course: sensing, movement, decision, and learning. So there would be four sections. There should be an introduction that basically describes what is in the section as well as indicates the role these materials will play in the Sense-Think-Act cycle. Ideally there will be a table of topics for the items in the section.

There can also be subsections. So if the section is movement, there would be sub-sections for path-finding and movement. Treat these as if they were sections when building the documentation.

Finally there are the specific techniques and procedures. Each of these should have a:

* Pattern name — everything needs a way to refer to it.
* Problem description — a description of the problem, its context, and when to apply the pattern. It might also include a list of requirements.
* Solution description — that describes the elements that make up the pattern and their relationships. This can be algorithm like. These should never be presented in a specific programming language.
* Discussion of Consequences — the results and trade-offs in using this pattern. Indicate the compromises that may be involved. This is critical for evaluating design alternatives.
* (optional) Sample implementation

As an example one might have a section called “movement” with a subsection called “path finding”. In the subsection description there would be a description of what is intended as well as a general abstract algorithm-like presentation. For example, you would want to include something like this:

find\_path (start\_node, end\_node):

open\_list = [start\_node]

closed\_list = []

while open\_list is not empty:

# Choose some node we know how to reach.

current\_node = choose\_node(reachable)

# If we just got to the goal node, build and return the path.

if current\_node == goal\_node:

return build\_path(goal\_node)

# Don't repeat ourselves.

open\_list.remove(current\_node)

closed\_list.add(current\_node)

# Where can we get from here that we haven't closed\_list before?

adjacent\_open\_list = get\_adjacent\_nodes(current\_node) – closed\_list

for adjacent\_node in adjacent\_open\_list:

# First time we see this node?

if adjacent\_node not in open\_list:

open\_list.add(adjacent\_node)

# If this is a new path, or a shorter path than what we have, keep it.

if current\_node.cost + 1 < adjacent\_node.cost:

adjacent\_node.previous = current\_node

adjacent\_node.cost = current\_node.cost + 1

# If we get here, no path was found

return None

Note that forgoing example could be easily condensed. Note that not all of the pieces needed for a specific pattern and implementation are present. The pieces that you would add would be in a pattern. The chose\_node function/method/procedure would indicate how it would be built for some general kind of problem. Under that would be the more specific details of how to build this in a problem context. There would, of course, be a discussion of the strengths and weaknesses of the pattern. You might want to add an implementation in a specific language. Check here for a bit of inspiration: <http://www3.cs.stonybrook.edu/~algorith/#>

So why should you want to do this? The answer is relatively simple. There is no set game environment and no set programming language. You want to have knowledge that you can use in context. If you would like to see the difference in environment and language styles, compare Construct 2 (<https://www.scirra.com/construct2> ) to Unity ( <https://unity3d.com/> ). So no matter what the environment and programming language, you will have the resources to begin your design work.