Final Year Project Report

Full Unit - Design Patterns for AI and Search

Playing Games and Solving Puzzles Using AI

Luke Sell

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Iddo Tzameret

Department of Computer Science
Royal Holloway, University of London

December 13, 2019

Declaration

Word Count: N/A

Student Name: Luke Sell

Date of Submission:

Signature: l.sell

Table of Contents

Theory

The following is some of the Design Patterns I found from reading [1] that may be relevant to my Program with a brief summary of their usefulness, as described in [1] , and an explanation of how I will implement them in my program.

Iterator

An iterator allows a way to access the elements of a data structure sequentially without knowing its underlying structure, this simplifies moving through a data structure as the code for knowing the next element is all in one place.

I could use this for the grid class, it stores a two dimensional array of entries that represent the grid of numbers, by using an iterator I would be able to get the next element in a one dimensional way without needing to know how to move from one row to the next, which can be used in the solver to work out which is the next square in the search.

Memento

A memento allows the internal state of an object to be saved and restored later, this means the values of an object fields are stored in the memento which is passed back to the object later when it needs to be restored.

This could be used to store the grid entries constraints, this would allow for the constraint solver to simulate the constraint propagation and reverse it easily if a later modification is not valid.

Observer

An observer is used to keep track of an object state, this way when an object changes state all of the objects observers can be notified of this, this allows a simplification of the relationships between the objects by decoupling them.

This could be used for grid entries and constraint propagation, when a grid entry is updated with a number all entries in the row, column and box can be notified of the new constraints from this number and update the possible numbers for that square appropriately.

State

Having states allows an object to change its behaviour when its internal state changes, this means one object can represent many different behaviours.

This could be used for grid entries, so that when they are selected or locked, their behaviour changes, i.e. a locked entry cannot be selected and a selected entry can be modified.

Strategy

A strategy allows algorithms that perform the same task to be grouped together and be used interchangeably, this would allow multiple algorithms to be designed and used in the program.

This could be used for the solver, the program would have multiple different algorithms that solve the Sudoku, but do it in different ways or more efficiently, this would allow for different optimizations such as dynamic variable ordering to be used and benchmarked against a basic solving algorithm.

Visitor

A visitor allows a different operation to be perform based on the type of object it will be performed on, this means an algorithm can be applied to a data structure which contains objects with different sub types.

This could be used in the solver to represent the different operations to be performed if the entry is an initial number on the grid or an empty square, which would simplify the solving algorithm structure.

Adapter

An adapter or wrapper converts the interface of a class to be usable by another class, i.e. allowing for an int to be added to a data structure that holds objects. This Design Pattern is often used in a similar way to the Facade Design Pattern.

This will be useful in my grid class to store the numbers on the grid as entry objects, while still allowing numbers to be passed as input.

Flyweight

A flyweight allows objects to be created for a highly used type, without using too much memory and time constructing the objects, this is best used when the amount of objects needed is small with each object being reused many times.

I could use this to represent numbers on the grid, i.e. instead of having numbers in an entry as a primitive I can make them into objects allowing for methods to be used on them, this works by creating an instance of each number from one to nine and storing them in a factory object, from which they can be returned. This way I can many of one number on the grid but only one object to represent it.

Bridge

A bridge decouples the abstraction from the implementation, this allows the two to vary independently, i.e. the implementation can be extended without changing the abstraction, this makes it easier to add new sub classes of a parent object.

I could use this to bridge to the correct implementation of a grid, depending on if the user selects Sudoku or Eight Queens or another puzzle, this would allow me to add different puzzles that sub class from the grid object without changing the code for the abstraction.

Bibliography

[1] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.*