Design Patterns in Action

Finding and Using Patterns in Software

Heurgame Framework

David Kaplin
NYU Masters Student
Overview

- Problem Domain (the Games!)
  - Voronoi
  - Nanomunchers
- Introduction to Design Patterns
- Patterns in Action (Heurgame Framework)
The Voronoi Game

- Each player has N stones to place on a grid
- Once a stone is placed it cannot be removed as long as the owner is in the game
- a Voronoi diagram is a tessellation of a plane into polygons
  - Every stone is in the interior of one polygon
  - For every point x in the polygon P associated with stone s, x is closer to s than it is to any other stone s'.
  - Distance is based on Euclidean distance.
- Play is turn based
- Largest area wins
All Players placed their first stones
Blue moved
Dark Green Moved
Dark Orange Moved
An advanced two player game...
Adversarial Nanomunchers

- You and your opponents each have 2 nanomunchers.
- Nanomunchers eat nodes. Nodes are connected via at most 4 edges.
- Nanomunchers move in a predetermined order \{Up, Down, Right, Left\}.
- If it is not possible to move in one direction the next one in its list is used.
- If there is no available node, a nanomuncher dies of starvation.
- If two nanomunchers advance on the same node the one with a better direction wins.
Nanomunchers

- At the start of the game all the information you have is how the nodes are connected
- You know nothing of your adversaries
- The only decisions you can make are
  - Where to place each nanomuncher
  - What program to run on each nanomuncher
Room For Variation

- What if alternating turns in Voronoi was not fair?
- What if you had three nanomunchers?
- What would change if you had eight directions instead of four in nanomunchers?

Abstraction minimizes these worries!
Patterns in Action
The Heurgame Framework

- **What is Heurgame?**
  A set of abstractions and utilities that should allow greater flexibility in building game software.
- **What Heurgame isn't**
  Heurgame is not a Game Engine!
Heurgame Framework Overview

- Game Logic
- Logging System
- Network System
- Event System
- User Interface
A Warning

There is no silver bullet in building software.

There is no single “right” way.
What is a Design Pattern?

“A Design Pattern is a description of communicating objects and classes that are customized to solve a general design problem in a particular context”
- Design Patterns, GoF

Design Patterns allow a commonly used shorthand for existing object-oriented structures.
Reading Patterns

- Unified Modeling Language (UML)
  - Visual representation of OO Systems
  - Class Diagram
  - Collaboration Diagram
  - and many others...
- Problem Domain
- Patterns associated
- Pros
- Cons
Common Design Patterns and where to find them

- Standard Library Patterns
  - Iterator
  - Observer
  - Factory
- User Interface Libraries
  - Command
  - State
  - Memento
  - Flyweight
- Compilers
  - Visitor
  - Composite
  - Builder
Networking System

- Designed to be protocol agnostic
- GameServer
- Referee
  - Deals with the Players via a PlayerProxy
- PlayerProxy
  - Uses the Proxy Pattern to allow both Socket Connected Players and Players directly interfacing with the server to be treated equally

Rest of Architecture → Referee

PlayerProxy “Bob” → Referee

PlayerProxy “Alice” → Referee

Python socket → Referee

Human → Referee
Proxy Pattern

- Use to restrict or control access to objects
- Relies on Delegation and Composition
- Also used where simple references are not enough
Game Logic

- Game – Provides the SystemAnalyzer
- SystemAnalyzer – Global Analysis
- MoveAnalyzer – Local Analysis
- Referee – Relies on the Move Analyzer
- Close to the Mediator Pattern
Mediator Pattern

- Encapsulates how objects interact
  - One Mediator
  - Many Colleagues
- Emphasis on loose coupling
Using Mediator

- Referee obtains a reference to the SystemAnalyzer.
- For each turn a TurnBasedReferee uses the MoveAnalyzer to see if the move is valid (out of bounds).
- The MoveAnalyzer in turn will query the SystemAnalyzer for wider conditions (duplicate move, illegal move).
- The only established relationship is that the System Analyzer has at least one Move Analyzer (who has a back reference).
Using Mediator

- There are no restrictions on the Number of System Analyzers or Move Analyzers
- More complex games may call for entire systems of classes to be involved.
  - Nanomunchers
    SystemAnalyzer knows
    a graph, the graph knows nodes
    a BotWrangler, that knows the bots
    the bots know where they can move
    the MoveAnalyzer talks to the SystemAnalyzer
Event System

- Anywhere there are events you will find the Observer Pattern
- Two Pieces
  - Subject – knows the Observers aka Broadcaster
  - Observer – Acts on the notification aka Listener
Observer Pattern

- Also known as Publish and Subscribe
- Very loose coupling is desired
- Observer is a very common pattern
Event System II

- TurnIterator generates a sequence of turn events in a particular order.
- This is a direct combination of the Subject of the Observer Pattern being crossed with the Iterator Pattern.
Hybrid Iterator-Observer

- Traditionally, Iterator spans an existing set of items
- Imagine a “Lazy Iterator” ... a Generator
The TurnIterator

- Isolates the Turn mechanism of the game
- Allows greater flexibility than a for loop
  - RandomTurn Generators
  - RoundRobinTurnIterator
    The standard way of iterating turns
  - SecondChanceRRTurnIterator
    In Voronoi the Second player is at a disadvantage, it can be proved that the first player can always win under strict alternation.

Solution: Give the second player two first turns.
User Interface

- Primarily on the Listener/Observer side of the Event System
- Examples:
  - ScoreBoard - Updates on PlayerEvents and TurnEvents
  - EstimatedTime – Updates on TimeEvents
- Complex UI will use the Builder Pattern (in development)
  - Three Parts: Director, Builder, Product
Builder Pattern

- Encapsulate the construction process while removing coupling to the actual product
Proposed Builder Pattern

- Abstract away mundane aspects of layout and certain repeated tasks
Logging System

- Loosely Coupled
- LogReaders, LogWriters, Log, DefaultLog
- Too much duplicated Code!
- Facade Pattern to the Rescue: LogBox
Facade

- Simple interface to complex system
Facade in Action
More Information

- Design Patterns: Elements of Resusable Object-Oriented Software
  - Authors “Gang of Four” (GoF)
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
  - http://hillside.net/patterns/DPBook/DPBook.html
- UML http://www.uml.org
- UML for Patterns
  http://www.tml.hut.fi/~pnr/Tik-76.278/gof/html/
- Heurgame Framework [BETA]
  - http://www.cs.nyu.edu/~dbk1/heurgame/
The Games

- Heurgame Implementations
- Voronoi
  - http://www.cs.nyu.edu/~dbk1/voronoi/
- Nanomunchers
  - http://www.cs.nyu.edu/~dbk1/nanomunchers/