Software Architectures
Assignment 1 - Game Engine

David O Neill 0813001
Gavin Fitzgerald 0857173

October 26, 2011

Marking Scheme

Table Of Contents

1 Introduction

1.1 Narrative

Design patterns are a general re-use solutions to commonly occurring context problems of software application development. Design patterns typically show the relationships between class instances at run time without specifying the final concrete implementations of the decoupled types. Furthermore it could be said that Design patterns are the layers between clients and the context of which they are designed to work with while acting as the glue that brings them together in an abstract fashion, allowing them to vary independently.

As part of Software Architectures we were tasked with the project scenario of a game system that implemented design patterns to supported the ilities or non functional requirements, some times otherwise known as quality attributes. Some of these quality attributes include, extensibility, modifiability and modularity and are the primary means of determining software quality. Software architectures which implement these aspirable software characteristics are said to be accomplished via the use of incorporating design patterns into the high level architecture.

These design patterns are categorized into varieties of creational, structural and behavioral patterns. As part of the project it was desirable to exhibit these categories through a small subset of the thousands of patterns which make up these categorical hierarchies. including Observer, Decorator, Strategy, Factory Method, Bridge and one self-researched pattern which we decided upon to be the memento pattern. These patterns lead into one another as shown in the diagram derived by the Gang of four, Gamma et al.

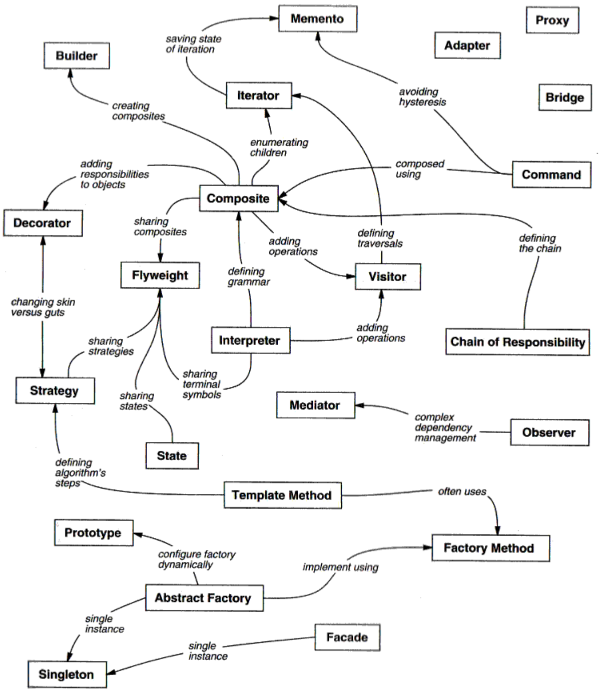


Figure 1.1: Design pattern relationships

Fig. 1.1: Gang of four - Pattern Relations

The patterns mentioned above were to implement the run time behaviour, in supporting notifications or events through the system, extending or changing objects as a means to avoid sub classing, behavioural strategies to complete tasks, creational patterns to allow the instantiation of different player types and teams, and the bridge pattern to allow variability between the different visual and text implementations of the game interface.

Before any code took place the design was essential to help visual the patterns and the interactions between these patterns and that in of the other supporting patterns in the engine. Visual paradigm was chosen to complete this task.

Throughout the development of the system constant refactoring took place to better realise the patterns in the formal sense, refactoring methods such as encapsulate field, generalize type, extract method and extract class where used to decouple and show the cohesive value of the implementation of the patterns.

2 Discussion - Patterns

2.1 Required Patterns

2.1.1 Observer

The observer pattern defines a one-to-many dependency between objects so that when one object changes state all of its dependents are notified about the change in state and update automatically. Creating highly coupled objects reduces their re-usability, so observer endeavors to maintain the consistency among the collection of these cooperating classes. The subject or the object that has changed state invokes changes on all the observers attached to it using the update method realized by the observer interface. This is done through the notify method in the observable interface.

As part of the community design, Observer in relation to Model View Controller would be used as a way to notify and update each clients views. In this case the database on the distributors system. A model adapter bound to the data source would receive an update via a notification as a result to the change in the model. Model changes would be propagate via the notification to a change to a view and the update operation on the model would be executed.

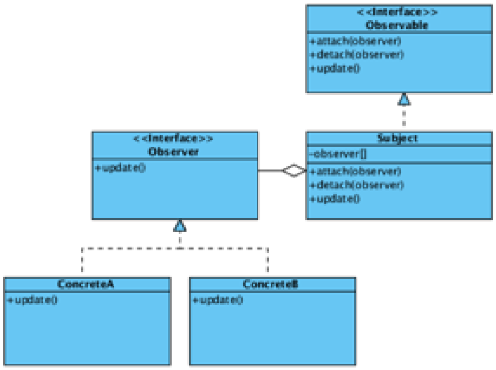


Fig. 2.1: Observer Design Pattern

2.1.2 Factory Method

The Factory Method Pattern defines an interface for constructing objects that lets subclasses decide which class to instantiate. It allows the user to defer class construction to a subclass which is needed when a class cant/shouldnt be able to know which class of objects it must construct i.e. the object construction ruses information or resources not appropriate for the composing object.

The Creator isnt bound to any Product class, it will work with any class that implements the Product Interface.

Can cause unnecessary sub classing e.g. we have Standard Team, and Three Team Creators for Three Player Products. Could have been avoided with templates or delegates.

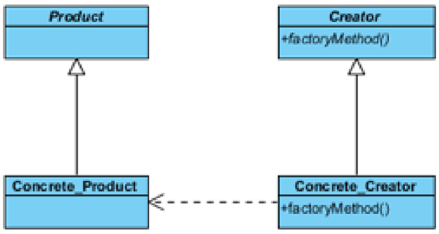


Fig. 2.2: Factory Method Design Pattern

2.1.3 Strategy

With the Strategy Pattern, we have different sets of behavior, similar in classification but different in implementation and we wish to make them interchangeable. It allows the behavior of an algorithm vary independently from the object that uses it.

Strategy allows us to group related algorithms but instead of having to subclass Context to give it different behaviors i.e. hard-wiring it, we can abstract that behavior to allow us to vary the algorithm dynamically. Strategy allow avoids bloated conditionals in deciding which behavior to use.

The drawbacks of Strategy include its coupling to the client, as the Client must be aware of the strategies to decide which strategy to use. Similarly, the strategy will likely need data from context so you must either pass context to the strategy (when it may not use all of Contexts data) or you may have to increase coupling between the context and strategies to avoid too much unnecessary data being passed.

In our case, Strategy worked well in allowed us to decide which AI moveset should be employed at run-time. It did force us however to pass the data of which selected AI down through many objects that did not need to be aware of such things, an unfortunate consequence.

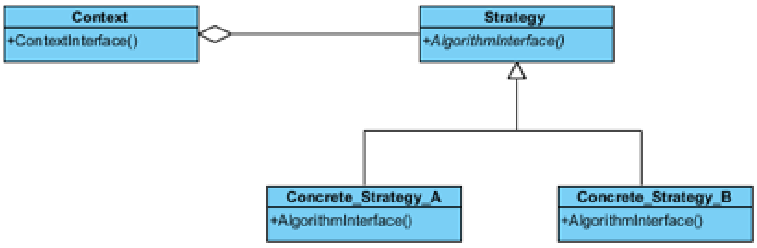


Fig. 2.3: Strategy Design Pattern

2.1.4 Bridge

The Bridge Pattern allows us to decouple abstraction and implementation so they be individually different. It is often considered to go beyond encapsulation and be considered insulation (http://www.vincehuston.org/dp/ insulation.html). Often when implementations are selected at run-time (what kind of display to use, in our projects case) the implementation ends up very tied to its abstraction, which an example of bad coupling and makes the product very complex. With Bridge we can change the implementation of the abstraction and have no impact on the client.

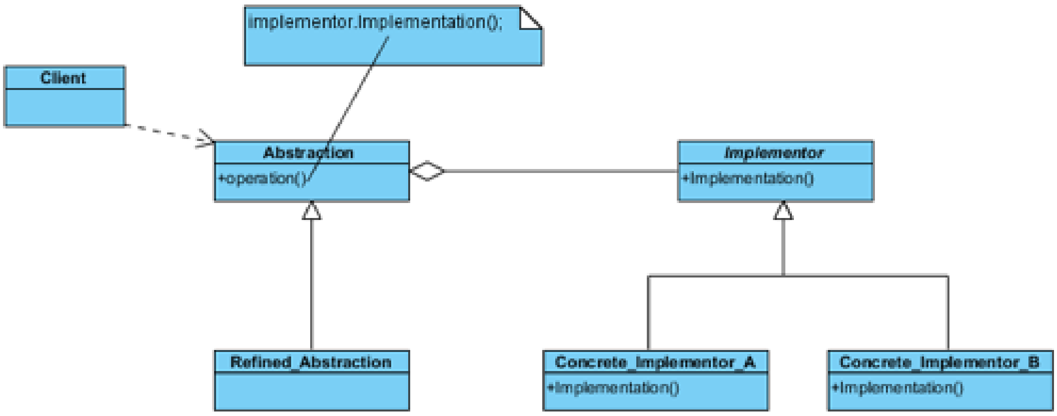


Fig. 2.4: Bridge Design Pattern

Obviously this allows us to extend Abstraction & Implementor independently, so weve greater extensibility and we have greater encapsulation because we can hide clients from implementation.

In our project, we use Builder to separate the usage of the user interface from how it is implemented. This allows us to have the abstracted functionality in the Game Window class, but have separate implementations of that functionality in Text_Window & Model_Window.

2.1.5 Decorator

The Decorator Pattern gives programmers an alternative to subclassing when wanting to add functionality to class, especially if one wishes to do it dynamically.Extending without subclassing is particularly handy when there are a large number of possible/likely extensions as subclassing would result in an enormous set of subclasses to support every combination. Decorator merely coats the existing object in one with the extended functionality.

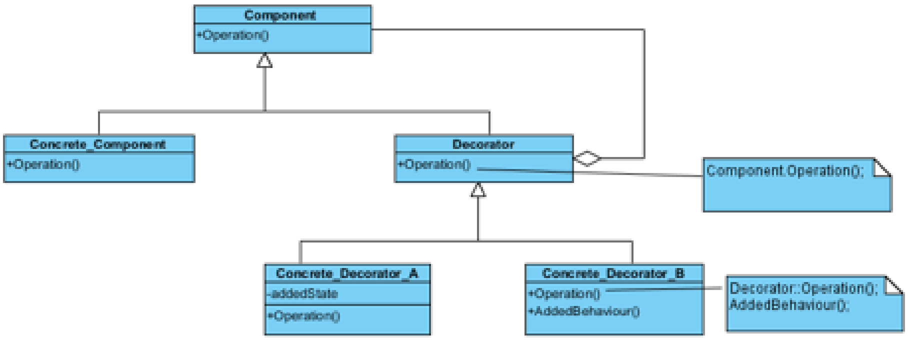


Fig. 2.5: Decorator Design Pattern

Decorator allows for dynamic growth of behavior rather than static inheritance.It also prevents use from subclassing every eventuality, applications dont need to pay for features they dont use. However Decorators are simply enclosures, a decorated component isnt the same as the component i.e. the object identity can be complicated, especially as another consequence of Decorator is that they will be many small objects with different pieces of functionality (rather than one super object).

In our project, Decorator is used to allow our concrete, or base map to have different kinds of obstacles on it e.g. mines. This avoids us subclassing many different kinds and combinations of maps e.g. MinesMap, LandOMap, MinesAndLandOMap etc.

2.2 Chosen Pattern

2.2.1 Memento

Without violating encapsulation the memento pattern captures and restores an objects internal state. The memento pattern creates check points, that let user back out or undo tentative operations.

An obvious candidate for saving state and returning to a state is the memento pattern. The ability to undo or rollback and action is advantageous not only to the end user but can reduce the load on a system. As part of the designing for concurrency and minimizing the network traffic, one of the design patterns considered was the memento pattern. By using the pattern the current state could be stored and restored later.

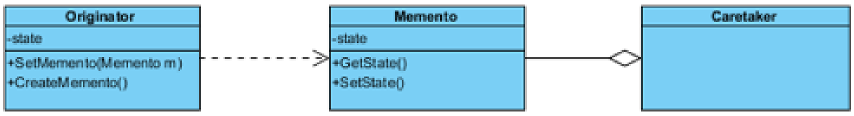


Fig. 2.6: Memento Design Pattern

By example when the user is browsing an online store, rather than re fetching the previous or next page the memento pattern can be used to capture and externalize the previous objects internal state thereby implementing an undo mechanism but also reducing network load and the interaction with the remote service. A checkout system is also a good candidate for this pattern as it would offer the user a rollback mechanism in the case of an error and the ability to recover from a mistake without having to redo the entire sequence of operation that led up to the undesired state.

Another benefit of Memento is the avoiding of creating a direct interface to get an objects state which could expose implementation details and thereby breaking encapsulation.

However, the Memento Pattern does have certain drawbacks. If the Originator has to copy large amounts of information to store in the memento and/or the client is creating and replacing mementos quite often then there will be considerable overhead that will likely affect performance and reliability. Generally one should avoid the Memento pattern unless encapsulating and restoring Originator states is cheap.

Similarly, depending on the size of the Memento, the caretaker might have a large storage cost when it stores mementos. On the issue of encapsulation, certain languages make it difficult to ensure only the originator can access the mementos state, being an issue with defining narrow/wide interfaces. By this I mean that the Caretaker sees a Narrow Interface (it can only pass Memento to other objects) while the Originator sees a wide interface, so that it can access all the data needed to restore itself to its previous state.

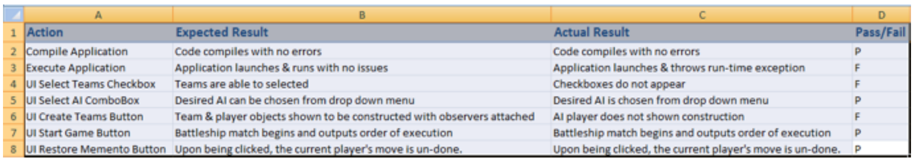
In our project Memento is used to allow the player to roll back after making a move. The Originator (the Player Object) has a state which is its ship positions, which changes after every move. Before moving, the Caretaker (the Game Object) asks the player to create a memento and stores it.

If after a player has moved (but during their turn) they click Undo, the Game Object will reset the state (in the Player Object) to the memento previously created and delete that memento from its collection of mementos.

3 Design

3.1 UML - Class Diagram

Fold out the next Page

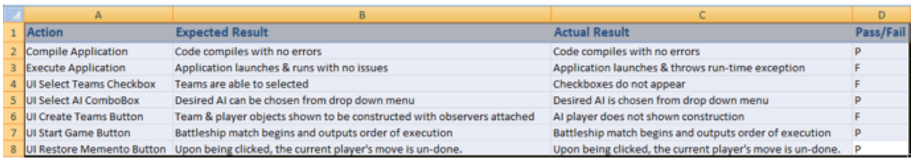


	A	B	C	D
1	Action	Expected Result	Actual Result	Pass/Fail
2	Compile Application	Code compiles with no errors	Code compiles with no errors	P
3	Execute Application	Application launches & runs with no issues	Application launches & throws run-time exception	F
4	UI Select Teams Checkbox	Teams are able to selected	Checkboxes do not appear	F
5	UI Select AI ComboBox	Desired AI can be chosen from drop down menu	Desired AI is chosen from drop down menu	P
6	UI Create Teams Button	Team & player objects shown to be constructed with observers attached	AI player does not shown construction	F
7	UI Start Game Button	Battleship match begins and outputs order of execution	Battleship match begins and outputs order of execution	P
8	UI Restore Memento Button	Upon being clicked, the current player's move is un-done.	Upon being clicked, the current player's move is un-done.	P

Fig. 3.1: Class Diagram

3.2 UML - Sequence Diagram

Fold out the next Page



Action	Expected Result	Actual Result	Pass/Fail
Compile Application	Code compiles with no errors	Code compiles with no errors	P
Execute Application	Application launches & runs with no issues	Application launches & throws run-time exception	F
UI Select Teams Checkbox	Teams are able to selected	Checkboxes do not appear	F
UI Select AI ComboBox	Desired AI can be chosen from drop down menu	Desired AI is chosen from drop down menu	P
UI Create Teams Button	Team & player objects shown to be constructed with observers attached	AI player does not shown construction	F
UI Start Game Button	Battleship match begins and outputs order of execution	Battleship match begins and outputs order of execution	P
UI Restore Memento Button	Upon being clicked, the current player's move is un-done.	Upon being clicked, the current player's move is un-done.	P

Fig. 3.2: Sequence Diagram

4 Development

4.1 Code

4.1.1 BattleshipTest.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Threading;
using System.Windows.Threading;

namespace Battleship
{
    /// <summary>
    /// Interaction logic for BattleshipTest.xaml
    /// </summary>
    public partial class BattleshipTest : Window
    {
        private static BattleshipTest instance;
        private ThreadStart threadstart;
        private Thread gameThread;
        private Game game;
        private int threadStatus = 0;

        private delegate void messageDelegate(string cname, string msg, Color color);
        private messageDelegate callBack;

        private List<string> selectedTeams = new List<string>();
        private List<string> computerAlgorithmTypes = new List<string>();

        #region Construction and destruction
        public BattleshipTest()
        {
            InitializeComponent();
            BattleshipTest.instance = this;
            this.callBack = this.addMessageToRichtextBox;
            this.richTextBox1.Document = new FlowDocument();

            string evolutionary = "Evolutionary Algorithm";
            string reinforcementAlg = "Reinforcement Algorithm";

            // Default Algorithms
            this.computerAlgorithmTypes.Add( evolutionary );
            this.computerAlgorithmTypes.Add( reinforcementAlg );

            // Check Box Filling & Default
            this.Computer_1_AI_cb.Items.Add( evolutionary );
            this.Computer_1_AI_cb.Items.Add( reinforcementAlg );

            this.Computer_1_AI_cb.SelectedIndex = 0;

            // Check Box Filling & Default
            this.Computer_2_AI_cb.Items.Add( evolutionary );
            this.Computer_2_AI_cb.Items.Add( reinforcementAlg );

            this.Computer_2_AI_cb.SelectedIndex = 0;

            this.game = new Game();

        }

        private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
        {
            try
            {
                this.gameThread.Abort();

                // While gameThread is terminating, main thread will wait for half a second
                System.Threading.Thread.Sleep( 500 );
            }
            catch( Exception )
            {

            }
        }
        #endregion

        #region Game control memento start game ui elements
        private void gameControlButton_Click(object sender, RoutedEventArgs e)
        {
            Console.WriteLine("called");
            try
            {
                if (threadStatus == 0)
                {
                    this.gameControlButton.Content = "Pause";
                    threadStatus = 1;
                    this.threadstart = new ThreadStart(game.StartGame);
                    this.gameThread = new Thread(threadstart);
                    this.gameThread.Start();
                }
                else if (threadStatus == 1)
                {
                    this.gameControlButton.Content = "Resume";
                    threadStatus = 2;
                    this.gameThread.Suspend();
                }
                else
                {
                    this.gameControlButton.Content = "Pause";
                    this.gameThread.Resume();
                }

            }
            catch (Exception)
            {
                MessageBox.Show("Unable to start the game thread");
            }
        }

        private void restoreMememtoButton_Click( object sender , RoutedEventArgs e )
        {
            this.game.UndoMove();
        }
        #endregion

        #region Team control ui elements
        private void createTeamsButton_Click( object sender , RoutedEventArgs e )
        {
            // Fowler would be upset, remove temps! bad code smell
            bool teamEasy = (bool) easyTeam_TB.IsChecked;
            bool teamHard = ( bool ) hardTeam_TB.IsChecked;
            bool teamComputer = ( bool ) computerTeam_TB.IsChecked;

            // Defensive code block, bad!
            if( teamEasy && teamHard && teamComputer )
            {
                MessageBox.Show( "Please select only two teams." );
                this.easyTeam_TB.IsChecked = false;
                this.hardTeam_TB.IsChecked = false;
                this.computerTeam_TB.IsChecked = false;
                this.selectedTeams.Clear();
            }
            else if( ( teamEasy && teamHard && !teamComputer )
                  || ( teamEasy && !teamHard && teamComputer )
                  || ( !teamEasy && teamHard && teamComputer ) )
            {
                if (teamEasy == true)
                {
                    this.selectedTeams.Add( "Easy" );
                }

                if ( teamHard == true)
                {
                    this.selectedTeams.Add( "Hard" );
                }

                if ( teamComputer == true)
                {
                    this.selectedTeams.Add( "Computer" );
                }

                this.computerAlgorithmTypes[ 0 ] =
                                           ( string ) this.Computer_1_AI_cb.SelectedValue;
                this.computerAlgorithmTypes[ 1 ] =
                                           ( string ) this.Computer_2_AI_cb.SelectedValue;

                this.game.CreateTeams( selectedTeams , computerAlgorithmTypes );
                this.gameControlButton.IsEnabled = true;
                this.restoreMememtoButton.IsEnabled = true;
            }
            else
            {
                MessageBox.Show("Please select 2 teams!");
            }
        }
        #endregion

        #region Message handling area for delgated messages to the UI
        public void addMessageToRichtextBox( string classname , string message , Color color )
        {
            string package;
            string objname;

            Paragraph p = new Paragraph();
            p.LineHeight = 5;

            int x = classname.LastIndexOf( ’.’ );

            if( x == -1 )
            {
                package = "";
                objname = classname;
            }
            else
            {
                package = classname.Substring( 0 , x );
                objname = classname.Substring( x + 1 , classname.Length - x - 1 );
            }

            Run r1 = new Run( "Package : " );
            r1.Foreground = new SolidColorBrush( Colors.Green );
            p.Inlines.Add( r1 );

            Run r2 = new Run( package );
            r2.Foreground = new SolidColorBrush( Colors.Red );
            p.Inlines.Add( r2 );

            Run r3 = new Run( "  Object : " );
            r3.Foreground = new SolidColorBrush( Colors.Green );
            p.Inlines.Add( r3 );

            Run r4 = new Run( objname );
            r4.Foreground = new SolidColorBrush( Colors.Red );
            p.Inlines.Add( r4 );

            Run r5 = new Run( "  Said : " );
            r5.Foreground = new SolidColorBrush( Colors.Green );
            p.Inlines.Add( r5 );

            Run r6 = new Run( message );
            r6.Foreground = new SolidColorBrush( color );
            p.Inlines.Add( r6 );

            this.richTextBox1.Document.Blocks.Add( p );
            this.richTextBox1.ScrollToEnd();
        }

        public static void acceptMessage( string classname , string message , Color color )
        {
            BattleshipTest.instance.addMessage( classname , message , color );
        }

        public void addMessage( string classname , string message , Color color )
        {
            this.Dispatcher.Invoke( System.Windows.Threading.DispatcherPriority.Background ,
                new System.Windows.Threading.DispatcherOperationCallback(
            delegate
            {
                this.callBack( classname , message , color );
                return null;
            } ) , null );
        }

        #endregion
    }
}
```

4.1.2 I User Interface.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;

namespace Battleship.GameEngine.User_Interface
{
    public interface I_User_Interface
    {
        void drawWindow( int dimensions );
        void drawShips( I_Ship[] ships );
        void drawObstacles(I_Map_Component[] obstacles);

    }

}
```

4.1.3 Game Window.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;

namespace Battleship.GameEngine.User_Interface
{
    public class Game_Window : I_Window
    {
        private I_User_Interface implementor;
        private Boolean typeOfWindow;

        public Game_Window(Boolean textWindow)
        {
            this.typeOfWindow = textWindow;

            BattleshipTest.acceptMessage(this.ToString(),
                "Constructed concrete window ", Colors.Purple);

            if (this.typeOfWindow == true)
            {
                this.implementor = new Text_Window();
            }
            else
            {
                this.implementor = new Model_Window();
            }
        }

        public void drawMap(I_Map_Component theMap)
        {
            this.implementor.drawWindow( theMap.getMapBoundary() );
            this.implementor.drawShips(theMap.getAllShips() );
            this.implementor.drawObstacles(theMap.getObstacleCoordinates());
        }

    }

}
```

4.1.4 Model Window.cs

```csharp
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;

namespace Battleship.GameEngine.User_Interface
{
    public class Model_Window : I_User_Interface
    {
        public Model_Window()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Purple);
        }

        public void drawWindow(int dimensions)
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "Drawing Model window", Colors.Purple);
        }

        public void drawShips(I_Ship[] ships)
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "Delegating to draw_Ships()", Colors.Purple);
            this.draw_Ships(ships);
        }

        public void drawObstacles(I_Map_Component[] obstacles)
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "Delegating to draw_Obsticles()", Colors.Purple);
            this.draw_Obstacles(obstacles);
        }

        public void draw_Ships(I_Ship[] ships)
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "drawing ship models", Colors.Purple);
        }

        public void draw_Obstacles(I_Map_Component[] obstacles)
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "drawing obsticle models", Colors.Purple);
        }

    }

}
```

4.1.5 Text Window.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;

namespace Battleship.GameEngine.User_Interface
{
    public class Text_Window : I_User_Interface
    {
        public Text_Window()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Purple);
        }

        public void drawWindow(int dimensions)
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "Drawing text window", Colors.Purple);
        }

        public void drawShips(I_Ship[] ships)
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "Delegating to draw_the_Ships()", Colors.Purple);
            this.draw_the_Ships(ships);
        }

        public void drawObstacles(I_Map_Component[] obstacles)
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "Delegating to draw_the_Obsticles()", Colors.Purple);
            this.draw_the_Obstacles(obstacles);
        }

        public void draw_the_Ships(I_Ship[] ships)
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "drawing ship text", Colors.Purple);
        }

        public void draw_the_Obstacles(I_Map_Component[] obstacles)
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "drawing obsticle text", Colors.Purple);
        }

    }

}
```

4.1.6 Game.cs

```
using System;
using System.Collections.Generic;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Collections;
using System.Windows.Media;
using System.Windows;

namespace Battleship.GameEngine.Boundary
{
    public class Game
    {
        private I_Team[] teams;
        private ArrayList mementos;
        private I_Window window;
        private I_Map_Component map;
        private Boolean gameOver;
        private int currentTeam = 0;
        private int currentPlayer = 0;
        private int counter = 4;
        private Player Player;
        private int undoMove = 0;

        public Game()
        {
            BattleshipTest.acceptMessage( this.ToString() , "Constructed" , Colors.Blue );

            this.mementos = new ArrayList();
            this.gameOver = false;
            this.teams = new Team[2];
            this.map = new Mine_Obstacle_Decorator(
                new Land_Obstacle_Decorator(new Concrete_Map_Component()));
            this.window = new Game_Window(true);
            this.window.drawMap(this.map);

        }

        public void CreateTeams(List<string> teamTypes, List<string> computerAlgorithmTypes)
        {
            // ** Hard-Coded Team Types **
            List<string> teamTypesDEBUG = new List<string>();
            teamTypesDEBUG.Add(teamTypes[0]);
            teamTypesDEBUG.Add(teamTypes[1]);


            I_Team stdTeam1 = new Standard_Team();
            this.teams[0] = stdTeam1.CreateTeams(teamTypesDEBUG[0], computerAlgorithmTypes);

            // ** Hard-Coded Passing of Game **
            I_Team stdTeam2 = new Standard_Team(this);
            this.teams[1] = stdTeam2.CreateTeams(teamTypesDEBUG[1], computerAlgorithmTypes);

            I_Player[] team1 = this.teams[0].GetPlayers();
            I_Player[] team2 = this.teams[1].GetPlayers();

            I_Player temp1 = team1[0];
            I_Player temp2 = team1[1];
            I_Player temp3 = team2[0];
            I_Player temp4 = team2[1];

            temp1.Attach(temp2);
            temp1.Attach(temp3);
            temp1.Attach(temp4);

            temp2.Attach(temp1);
            temp2.Attach(temp3);
            temp2.Attach(temp4);

            temp3.Attach(temp1);
            temp3.Attach(temp2);
            temp3.Attach(temp4);

            temp4.Attach(temp1);
            temp4.Attach(temp2);
            temp4.Attach(temp3);
        }

        public void StartGame()
        {
            this.teams[0].Attach( this.teams[1] );
            this.teams[1].Attach( this.teams[0] );

            while (!this.gameOver)
            {
                if( this.undoMove == 1 )
                {
                    this.Player.SetMemento(
                            ( Memento ) this.mementos[ this.mementos.Count - 1 ] );
                    this.mementos.Remove( this.mementos.Count - 1 );
                    this.undoMove = 0;
                }

                if( this.counter % 4 == 0 )
                {
                    this.currentTeam = 0;
                    this.currentPlayer = 0;
                }
                else if( this.counter % 4 == 1 )
                {
                    this.currentTeam = 1;
                    this.currentPlayer = 0;
                }
                else if( this.counter % 4 == 2 )
                {
                    this.currentTeam = 0;
                    this.currentPlayer = 1;
                }
                else if( this.counter % 4 == 3 )
                {
                    this.currentTeam = 1;
                    this.currentPlayer = 1;
                }

                I_Player[] players = this.teams[ this.currentTeam ].GetPlayers();
                this.Player = ( Player ) players[ this.currentPlayer % 2 ];

                this.mementos.Add( this.Player.CreateMemento() );

                BattleshipTest.acceptMessage( this.Player.ToString() ,
                                                "make move" , Colors.Blue );
                this.Player.Notify( "Fired" );

                this.Player.makeMove();

                this.counter++;

                System.Threading.Thread.Sleep(5000);
            }
        }

        public I_Team[] GetTeams()
        {
            return this.teams;
        }

        public void SetTeams(Team[] teams)
        {
            this.teams = teams;
        }

        public void UndoMove()
        {
            if( this.mementos.Count > 0 )
            {
                this.undoMove = 1;
                this.counter--;
            }
        }
    }

}
```

4.1.7 I Observer.cs

```csharp
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;

namespace Battleship.GameEngine
{
    public interface I_Observer
    {
        void Notify( string message );
        void Update( I_Observer subject , String message );
        void Attach( I_Observer I_Observer );
        void Detach( I_Observer I_Observer );

    }

}
```

4.1.8 I Team.cs

```
using System;
using System.Windows.Media;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;

namespace Battleship.GameEngine.Group
{
    public interface I_Team : I_Observer
    {
        I_Player[] GetPlayers();
        void SetPlayers( I_Player[] players );
        int GetNumPlayers();
        Color GetColor();
        void SetColor( Color color );
        I_Team CreateTeams(string typeOfTeam,
            System.Collections.Generic.List<string> computerAlgorithmTypes);
    }

}
```

4.1.9 Standard Team.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;
using System.Windows;

namespace Battleship.GameEngine.Group
{
    public class Standard_Team : Team
    {
        protected Game game;

        public Standard_Team()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
        }

        public Standard_Team( String type )
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
        }

        public Standard_Team( Game game )
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
            this.game = game;
        }

        public override I_Team CreateTeams(
            string typeOfTeam, List<string> computerAlgorithmTypes)
        {

            switch( typeOfTeam )
            {
                default:
                    return this;

                case "Easy":
                    return new Easy_Team();

                case "Hard":
                    return new Hard_Team();

                case "Computer":
                    // ** Hard Coded Passing of Game
                    return new Computer_Team(computerAlgorithmTypes);
            }
        }

        public override I_Player[] GetPlayers()
        {
            Console.WriteLine( "Called" );
            Console.WriteLine( this.Players.Length );
            return this.Players;
        }

        public override void SetPlayers( I_Player[] players )
        {
            this.Players = players;
        }

        public override int GetNumPlayers()
        {
            return this.Players.Length;
        }

        public override Color GetColor()
        {
            return this.color;
        }

        public override void SetColor( Color color )
        {
            this.color = color;
        }
    }
}
```

4.1.10 Team.cs

```
using System;
using System.Windows.Media;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using Battleship.GameEngine;
using System.Collections.Generic;
using System.Collections;


namespace Battleship.GameEngine.Group
{
    public abstract class Team : I_Team
    {
        protected I_Player[] Players;
        protected int TeamScore;
        protected Game Game;
        protected Color color;
        protected ArrayList observers;

        public Team()
        {
            this.observers = new ArrayList();
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
        }

        public Team( String type )
        {
            this.observers = new ArrayList();
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
        }

        public Team( Game game )
        {
            this.observers = new ArrayList();
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
        }

        public abstract I_Player[] GetPlayers();
        public abstract void SetPlayers(I_Player[] players);
        public abstract int GetNumPlayers();
        public abstract Color GetColor();
        public abstract void SetColor( Color color );
        public abstract I_Team CreateTeams(
            string typeOfTeam, List<string> computerAlgorithmTypes);

        public void Notify( string message )
        {
            foreach( I_Observer obj in this.observers )
            {
                obj.Update( this , message );
                BattleshipTest.acceptMessage(this.ToString(),
                    "Sending messsage to observer", Colors.Orange);
            }
        }

        public void Update( I_Observer subject , String message )
        {
            BattleshipTest.acceptMessage( this.ToString() ,
                "Recevied : \"" + message + "\" from : " + subject.ToString() , Colors.Orange);
        }

        public void Attach( I_Observer observer )
        {
            this.observers.Add( observer );
            BattleshipTest.acceptMessage(this.ToString(),
                "Observer was attached to me", Colors.Orange);
        }

        public void Detach( I_Observer observer )
        {
            this.observers.Remove( observer );
        }
    }
}
```

4.1.11 Hard Team.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;

namespace Battleship.GameEngine.Group
{
    public class Hard_Team : Standard_Team
    {
        public Hard_Team()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
            this.Players = new Player[ 2 ];
            this.Players[ 0 ] = new Hard_Player();
            this.Players[ 1 ] = new Hard_Player();
            Console.WriteLine( "Created players" );
        }
    }

}
```

4.1.12 Easy Team.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;

namespace Battleship.GameEngine.Group
{
    public class Easy_Team : Standard_Team
    {
        public Easy_Team()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
            this.Players = new Player[ 2 ];
            this.Players[ 0 ] = new Easy_Player();
            this.Players[ 1 ] = new Easy_Player();
            Console.WriteLine( "Created players" );
        }

    }

}
```

4.1.13 Computer Team.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;
using System.Windows;

namespace Battleship.GameEngine.Group
{
    public class Computer_Team : Standard_Team
    {
        public Computer_Team()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
            this.Players = new Player[2];
            this.Players[0] = new Computer();
            this.Players[1] = new Computer();
            Console.WriteLine("Created players");
        }

        public Computer_Team(string x)
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
            this.Players = new Player[2];


            Console.WriteLine("Created players");
        }

        public Computer_Team(System.Collections.Generic.List<string> computerAlgorithmTypes)
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
            this.Players = new Player[2];
            this.Players[0] = new Computer(computerAlgorithmTypes[0]);
            this.Players[1] = new Computer(computerAlgorithmTypes[1]);


            Console.WriteLine("Created players");
        }

        public override I_Player[] GetPlayers()
        {
            Console.WriteLine(this.Players.Length);
            return this.Players;
        }
    }

}
```

4.1.14 I Ship.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;

namespace Battleship.GameEngine.Gamepiece
{
    public interface I_Ship
    {
        int GetEndX();
        int GetEndY();
        void GetOwner();

        int GetStartX();
        int GetStartY();
        void SetEndX( int endX );

        void SetEndY( int endY );
        void SetStartX( int startX );
        void SetStartY( int startY );
    }

}
```

4.1.15 Ship.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;

namespace Battleship.GameEngine.Gamepiece
{
    public class Ship : I_Ship
    {
        private int startX;
        private int startY;
        private int endX;
        private int endY;
        private bool destroyed;
        private String owner;
        private I_Player i_Player;

        public Ship( object i_Player_owner )
        {

        }
        public int GetStartX()
        {
            return this.startX;

        }
        public void SetStartX( int startX )
        {

        }
        public int GetStartY()
        {
            return this.startY;

        }
        public void SetStartY( int startY )
        {

        }
        public int GetEndX()
        {
            return this.endX;

        }
        public void SetEndX( int endX )
        {

        }
        public int GetEndY()
        {
            return this.endY;

        }
        public void SetEndY( int endY )
        {

        }
        public void GetOwner()
        {

        }


    }

}
```

4.1.16 I Player.cs

```
using System;
using Battleship.GameEngine;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;

namespace Battleship.GameEngine.Participant
{
    public interface I_Player : I_Observer
    {
        I_Ship[] GetMyShips();
        int GetNumShips();
        void SetMyShips( I_Ship[] ships );

        void SetNumShips( int numShips );
        String GetName();

        void SetName( String name );
        int GetPower();
        void SetPower( int power );

        int GetPlayerScore();
        void SetPlayerScore( int score );

        void makeMove();


    // method detach is inherited from base class
    // method attach is inherited from base class
    // method update is inherited from base class
    // method notify is inherited from base class
    }

}
```

4.1.17 Player.cs

```
using System;
using Battleship.GameEngine;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Collections;
using System.Windows.Media;

namespace Battleship.GameEngine.Participant
{
    public class Player : Originator , I_Player
    {
        private I_Ship[] myShips;
        private int myShipsRemaining;
        private string name;
        private int power;
        private int playerScore;
        private I_Ship i_Ship;
        private Team team;
        private ArrayList observers;

        public Player()
        {
            this.observers = new ArrayList();
            this.myShips = new Ship[ 5 ];
        }

        public Memento CreateMemento()
        {
            BattleshipTest.acceptMessage( this.ToString() ,
                "Created memento of my ships" , Colors.Lime );
            return new Memento( this.myShips );
        }

        public void SetMemento( Memento memento )
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "restoring memento of my ships", Colors.Lime);
            this.myShips = memento.GetState();
        }

        public void Notify( string message )
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "Sending messsage to observers", Colors.Orange);
            foreach( I_Observer obj in this.observers )
            {
                obj.Update( this , message );
            }
        }

        public void Update( I_Observer subject , String message )
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "Recevied : \"" + message + "\" from : " + subject.ToString(), Colors.Orange);
        }

        public void Attach( I_Observer observer )
        {
            this.observers.Add( observer );
            BattleshipTest.acceptMessage(this.ToString(),
                "Observer was attached to me", Colors.Orange);
        }

        public void Detach( I_Observer observer )
        {
            this.observers.Remove( observer );
        }

        public I_Ship[] GetMyShips()
        {
            return this.myShips;
        }

        public void SetMyShips( I_Ship[] myShips )
        {
            this.myShips = myShips;
        }

        public int GetNumShips()
        {
            return this.myShips.Length;
        }

        public void SetNumShips( int myShipsRemaining )
        {
            this.myShipsRemaining = myShipsRemaining;
        }

        public string GetName()
        {
            return this.name;
        }

        public void SetName( string name )
        {
            this.name = name;
        }

        public int GetPower()
        {
            return this.power;
        }

        public void SetPower( int power )
        {
            this.power = power;
        }

        public int GetPlayerScore()
        {
            return this.playerScore;
        }

        public void SetPlayerScore( int playerScore )
        {
            this.playerScore = playerScore;
        }

        public virtual void makeMove()
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "human player made move", Colors.Blue);
        }

    }
}
```

4.1.18 Hard Player.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;

namespace Battleship.GameEngine.Participant
{
    public class Hard_Player : Player
    {
        public Hard_Player()
        {
            BattleshipTest.acceptMessage( this.ToString() , "Constructed" , Colors.Magenta );
        }
    }
}
```

4.1.19 Easy Player.cs

```csharp
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;

namespace Battleship.GameEngine.Participant
{
    public class Easy_Player : Player
    {
        private Easy_Team easy_Team;

        public Easy_Player()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
        }
    }
}
```

4.1.20 Computer Player.cs

```
using System;
using System.Windows.Media;

using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;


namespace Battleship.GameEngine.Participant
{
    public class Computer : Player
    {
        private string moveStrategyName;
        private Move_Strategy moveStrategy;

        public Computer()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Magenta);
        }

        public Computer(string moveStrategyName)
        {
            this.moveStrategyName = moveStrategyName;
            setMoveStrategy();
        }

        private void setMoveStrategy()
        {
            switch (moveStrategyName)
            {
                default:
                    moveStrategy = new Evolutionary_Move_Strategy();
                    break;

                case ("Evolutionary Algorithm"):
                    moveStrategy = new Evolutionary_Move_Strategy();
                    break;

                case ("Reinforcement Algorithm"):
                    moveStrategy = new RL_Move_Strategy();
                    break;
            }
        }

        public override void makeMove()
        {
            this.moveStrategy.AlgorithmInterface();
        }

    }

}
```

4.1.21 Move Strategy.cs

```
using System;
using System.Windows.Media;

using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;

namespace Battleship.GameEngine.Participant
{
    public abstract class Move_Strategy
    {
        public Move_Strategy()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Navy);
        }

        public abstract void AlgorithmInterface();
    }

}
```

4.1.22 RL Move Strategy.cs

```
using System;
using System.Windows.Media;

using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;


namespace Battleship.GameEngine.Participant
{
    public class RL_Move_Strategy : Move_Strategy
    {
        public RL_Move_Strategy()
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "Constructing Reinforcement Learning Alg for Computer Player", Colors.Navy);
        }

        public override void AlgorithmInterface()
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "made Reinforcement learning move", Colors.Navy);
        }
    }

}
```

4.1.23 Evolutionary Move Strategy.cs

```
using System;
using System.Windows.Media;

using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;


namespace Battleship.GameEngine.Participant
{
    public class Evolutionary_Move_Strategy : Move_Strategy
    {
        public Evolutionary_Move_Strategy()
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "Constructing Evolutionary Alg for Computer Player", Colors.Navy);
        }

        public override void AlgorithmInterface()
        {
            BattleshipTest.acceptMessage(this.ToString(),
                "made evolutionary learning move", Colors.Navy);
        }
    }

}
```

4.1.24 I Map Component.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;

namespace Battleship.GameEngine.Maze
{
    public interface I_Map_Component
    {
        I_Ship[] getAllShips();

        int getMapBoundary();

        I_Map_Component[] getObstacleCoordinates();

        void Populate_Map();
    }

}
```

4.1.25 Concrete Map Component.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;

namespace Battleship.GameEngine.Maze
{
    public class Concrete_Map_Component : I_Map_Component
    {
        protected I_Ship[] allShips;
        protected int mapBoundary;
        protected I_Map_Component[] obstacleCoordinates;

        public Concrete_Map_Component()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Brown);
        }

        public void Populate_Map()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Populated", Colors.Brown);
        }

        public I_Ship[] getAllShips()
        {
            return this.allShips;

        }

        public int getMapBoundary()
        {
            return this.mapBoundary;
        }

        public I_Map_Component[] getObstacleCoordinates()
        {
            return this.obstacleCoordinates;
        }

    }

}
```

4.1.26 Map Decorator.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;

namespace Battleship.GameEngine.Maze
{
    public abstract class Map_Decorator : I_Map_Component
    {
        protected I_Map_Component map;
        protected I_Ship[] allShips;
        protected int mapBoundary;
        protected I_Map_Component[] obstacleCoordinates;

        public Map_Decorator()
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Brown);
        }

        public Map_Decorator( I_Map_Component component )
        {
            BattleshipTest.acceptMessage(this.ToString(), "Constructed", Colors.Brown);
            this.map = component;
        }

        public virtual void Populate_Map()
        {
            if (map != null)
            {
                map.Populate_Map();
            }
        }

        public I_Ship[] getAllShips()
        {
            return this.allShips;

        }

        public int getMapBoundary()
        {
            return this.mapBoundary;
        }

        public I_Map_Component[] getObstacleCoordinates()
        {
            return this.obstacleCoordinates;
        }

    }

}
```

4.1.27 Mine Obstacle Decorator.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;

namespace Battleship.GameEngine.Maze
{
    public class Mine_Obstacle_Decorator : Map_Decorator
    {
        public Mine_Obstacle_Decorator(I_Map_Component comp) : base(comp)
        {
        }

        public override void Populate_Map()
        {
            base.Populate_Map();

            BattleshipTest.acceptMessage(this.ToString(),
                "Populated with mines.", Colors.Brown);
        }

    }

}
```

4.1.28 Land Obstacle Decorator.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;
using System.Windows.Media;

namespace Battleship.GameEngine.Maze
{
    public class Land_Obstacle_Decorator : Map_Decorator
    {
        public Land_Obstacle_Decorator(I_Map_Component comp) : base(comp)
        {
        }

        public override void Populate_Map()
        {
            base.Populate_Map();

            BattleshipTest.acceptMessage(this.ToString(),
                        "Populated with land obstacles.", Colors.Brown);
        }

    }

}
```

4.1.29 Originator.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;

namespace Battleship.GameEngine
{
    public interface Originator
    {
        Memento CreateMemento();
        void SetMemento(Memento memento);
    }

}
```

4.1.30 Memento.cs

```
using System;
using Battleship.GameEngine.Boundary;
using Battleship.GameEngine.Gamepiece;
using Battleship.GameEngine.Maze;
using Battleship.GameEngine.Participant;
using Battleship.GameEngine.Group;
using Battleship.GameEngine.User_Interface;

namespace Battleship.GameEngine
{
    public class Memento
    {
        private I_Ship[] state;
        private Player player;

        public Memento(I_Ship[] state)
        {
        }

        public I_Ship[] GetState()
        {
            return this.state;
        }

    }

}
```

5 Testing

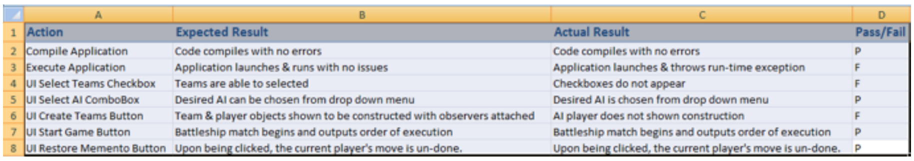
For our program, we felt given the size of the project, that it lent itself to smoke testing. The small nature of the project (and most college projects) meant that the core functionality of the project, is largely the entire functionality.

We werent implementing game logic, merely architectural logic which ties very neatly to smoke testing, where one does quick & dirty tests that test the major functions of a piece of software work. Smoke testing is a methodology that ”originated in the hardware testing practice of turning on a new piece of hardware for the first time and considering it a success if it does not catch on fire”.

So after every change, we would recompile, re-run through the project, and providing we introduced no new issues, we would move on to the next feature. As the book ”Lessons Learned in Software Testing” [3] puts it, ”smoke tests broadly cover product features in a limited time ... if key features don’t work or if key bugs haven’t yet been fixed, your team won’t waste further time installing or testing”.

This was assisted by the specification which required us to print out the methods and show our patterns at work. This meant that we had a clear chart of the order of execution of our program and immediately highlighted any issues. The customization afforded to us by Windows Presentation Foundation made such interface issues very easy to grapple with.

We largely felt that black-box testing suited our needs for the project, especially if we were to go and implement more game-logic at a later stage. This resulted in us making a simple set of test cases that we could quickly run through, especially in the later stages of our project :



Action	Expected Result	Actual Result	Pass/Fail
Compile Application	Code compiles with no errors	Code compiles with no errors	P
Execute Application	Application launches & runs with no issues	Application launches & throws run-time exception	F
UI Select Teams Checkbox	Teams are able to selected	Checkboxes do not appear	F
UI Select AI ComboBox	Desired AI can be chosen from drop down menu	Desired AI is chosen from drop down menu	P
UI Create Teams Button	Team & player objects shown to be constructed with observers attached	AI player does not shown construction	F
UI Start Game Button	Battleship match begins and outputs order of execution	Battleship match begins and outputs order of execution	P
UI Restore Memento Button	Upon being clicked, the current player's move is un-done.	Upon being clicked, the current player's move is un-done.	P

Fig. 5.1: Simple test cases

6 Critique & Discussion

It became apparent very early on that the use of so many patterns in such a small compact framework would lead to implementation issues arising in the form of poor realization of requirements and the resulting scope of the application. Since the project was primarily focused on realising the patterns, the business concerns were somewhat self evident but they were not the primary focus of the application and as a result the patterns did not contiguously flow together as expected.

After the initial phase of design the team set about implementing the code. During this phase we realized that even though we had successfully compartmentalized the patterns to best show their functionality, key information required for the interaction and between them was missing, such as references to the calling classes, how information for the user interface would be delegated to or passed to the concrete classes.

We set about refactoring the code to provide this delegation of data from the top level of the program to the inner workings of the classes via either constructors, accessors and mutators. It was import also to account for the fluidness of the application while keeping the constructs of the patterns in tact in order to present this in the form expected for the project.

On retrospect this could have been accomplished via boundary and control classes as would be expected in good Object Oriented Analysis and Design however as this was not done but could have been accomplished using many of the requirements analysis techniques such as noun identification technique.

As novices in the field of software engineering, we believe as a team that academic research and learning in the aid of realizing a software architecture can only bring it so far. From a novice perspective the experience required in the design of software architectures can only be accomplished by doing and learning; to the point that Extreme programming would have better realized the implementation.

Via coding, testing, listening designing the architecture could have been better implemented from a under graduate perspective. Through pair programming as seen in the lab exercises, shorter iterations, of designing, coding and listening together with an iterative waterfall approach would have resulted in a better overall project. This implication would have resulted in greater overall understanding, which could potentially lead to us, as undergraduates being better software architects from a design point of view.

Some Benefits we identified :

Every difficulty encountered in software engineering has been approached and usually design patterns have been implemented to solve the intent of these problems. That said design patterns when researched and understood by all parties in a software engineering project can help facilitate the rapid design and problem solving of solutions.

These proven practices and methodologies allow developers to exploit the experience of their predecessors increasing the productivity and understandability of software in the field and how it should be designed. Using these pattern methodologies allow for the extensibility and resusabilty of software systems in varying circumstances.

As mentioned in the narrative, the primary gain of patterns are the ilities or quality attributes, after all software reusability is key to market success, if it reusable then we need not reinvent the wheel again as the solution is already either already accessible or their is a solution which can be easily modified or extended to cater for the functionality we wish for it to provide.

Some Liabilities we identified :

Design Patterns are not without liabilities, or at least consequences one should keep in mind before using them. These "liabilities" include increased complexity, code bloat & a certain level risk of outdated application.

In traditional (small) college projects, design patterns rarely appear, apart from when thats a requirement of the project. This is because (beside ignorance of design patterns) the use of design patterns in smaller projects tends to introduce lots of unnecessary complexity and refactoring to mold an easy more-procedural esc application to a design-pattern, oo-heavy one. The amount of code bloat in this over-design out ways the benefits of design patterns for smaller scale projects.

Certain patterns fair better than others as time has gone by. Singleton can be compared to a global variable in disguise, which in an OO world is considered a mistake. The GoF book, while great perhaps should not completely be learned, but learned from, as its practically in its 36th print of its First Edition, a lot of learning has surely occurred in the past 20 years.

Bibliography

[1] *Refactoring.* Addison Wesley.

[2] *Design Patterns : Elements of reusable object orientated softwarePatterns of reusable object oriented software.* Addison Wesley, 1997.

[3] Bach Kaner and Pettichord. *Lessons Learned in Software Testing.* Wiley Computer Publishing, 2002.

[4] Meyers. Bridge pattern = insulation.

[5] Software testing specialists. Definition of smoke testing.

List of Figures