# IMC ASSIGNMENT SOLUTION

JUNE 4, 2019

## CONTENTS

## ASSIGNMENT PROBLEMS

**Problem 1. Visitor**

- Create an interface or abstract base class 'Shape'.
- Implement three concrete Shapes: 'Circle', 'Rectangle', 'Triangle'.
- Apply the visitor pattern to your data model.
- Write an AreaVisitor that computes the area of the shapes.
- Write a program that makes arbitrary instances of the data model and applies the AreaVisitor.

**Problem 2. Paper Scissors Rock**

- Paper Scissors Rock is a game for two players. Each player simultaneously opens his/her hand to display a symbol:
    - Fist equals rock,
    - Open hand equals paper,
    - Showing the index and middle finger equals scissors.
- The winner is determined by the following schema:
    - paper beats (wraps) rock,
    - rock beats (blunts) scissors,
    - scissors beats (cuts) paper.
- Write a program that plays Paper Scissors Rock between the computer and a real player.
- You should be able to play the game n times before the program exits.

## ASSIGNMENT SOLUTION

**It consists of following key modules**

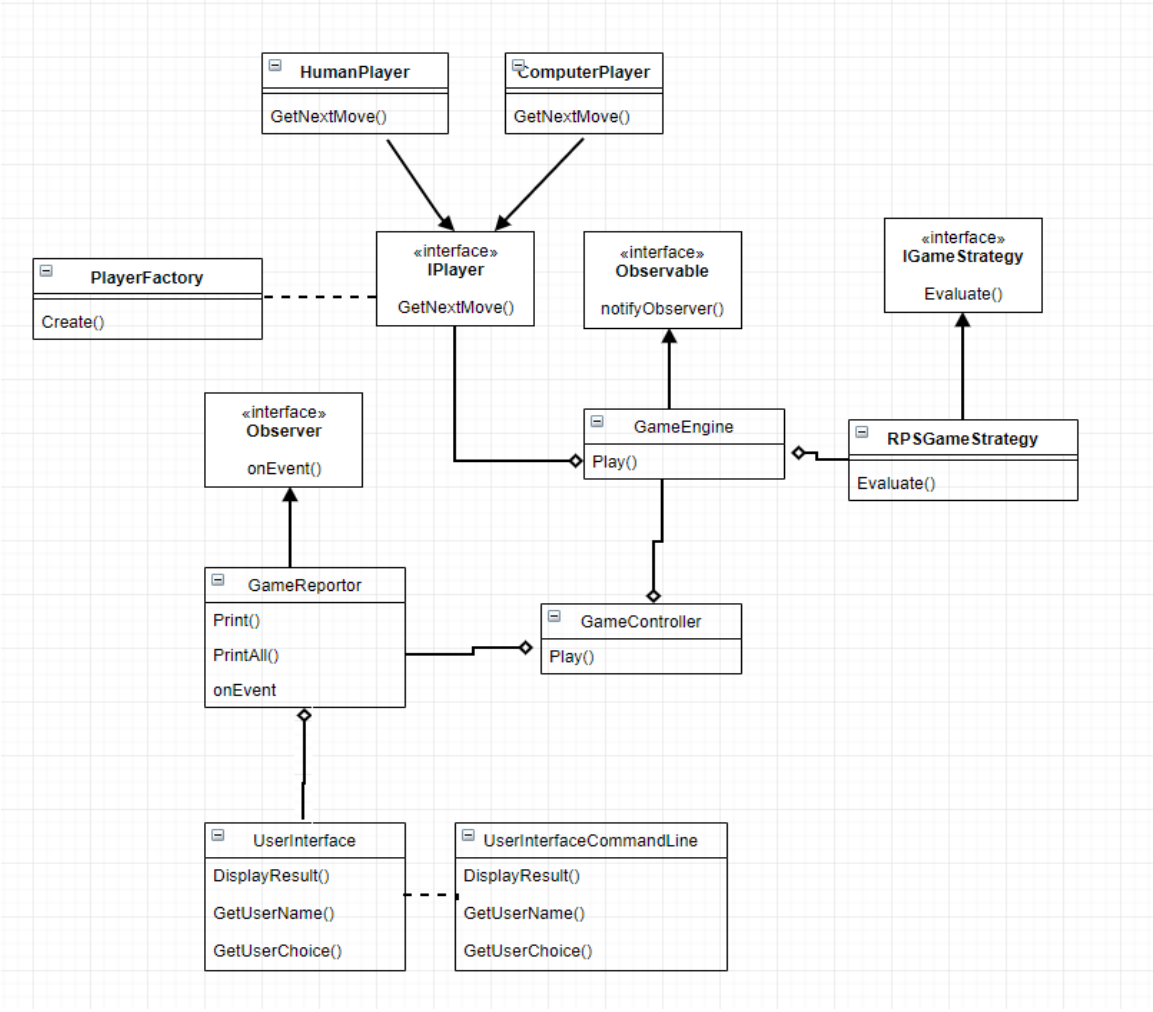
1. **RPS –** Solution files for Paper Scissors Rock game. Key modules include GameController, GameEngine, Player, PlayerFactory and UserInterface.
2. **Visitor –** 3 Variants of Visitors for Shapes hierarchy for AreaVisitor
    a. CompileTimeVisitorSimple
    b. RunTimeVisitorAcyclic
    c. RunTimeVisitorCyclic
3. **Main –** Client for RPS and Visitor Code
4. **Tester –** Unit tests for RPS and Visitor Code

## RPS LIB - TASK 2

1. **GameController(Singleton)**: Controller class for RPS Game Engine. This class can be extended to use multithreading (if we have future NW based clients) with every thread running own Play function.
2. **GameEngine (Observable)** - Engine to play the RPS Game between 2 Players. Human/Computer. It is observed by Game Reporter for reporting purposes.
3. **IStrategy/GameStrategy** – Strategy/Rules for the game
4. **IPlayer** : Abstract base class for Player which is specialized for
    a. HumanPlayer – Decides the next move by taking input from Human User
    b. ComputerPlayer - Decides the next move by taking input from Computer. Internally it uses random Function to take the decision
5. **PlayerFactory** : Factory to help with heirarchy of Player Objects.
6. **GameReporter(Observer)**: Report for Gameoutcomes and print overall report
7. **UserInterface** : Class to interface UserInterface. It aims to plug the different UserInterface (UICommandLine, or NW based in coming time.) to GameEngine

**Design Patterns used** – Singleton, Observer, Factory, Strategy.

## CLASS DIAGRAM FOR RPS – (ROCK PAPER SCISSOR GAME)



## VISITOR LIB – TASK 1

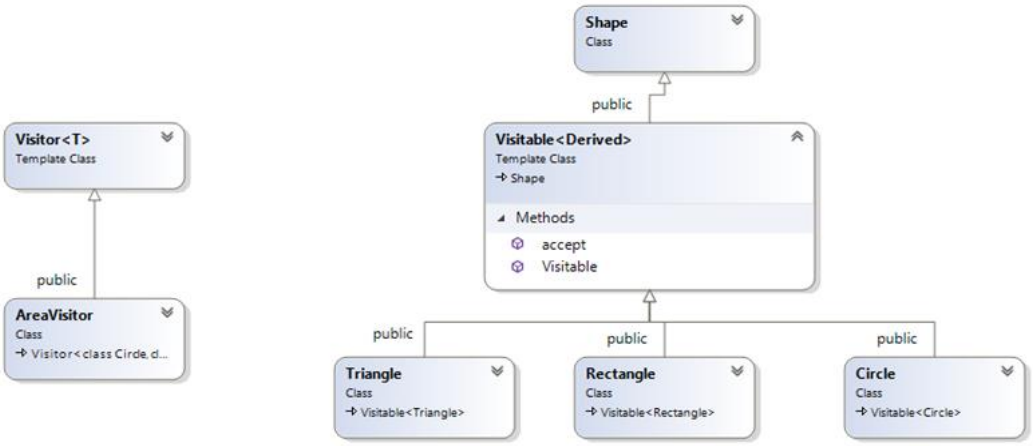
It elaborates, with few variations the usage of Visitor pattern which effectively allows us to add a new virtual function to the entire class hierarchy without changing the source code of the classes. The hierarchy must be made visitable, but after that, any number of operations can be added, and their implementation is kept separate from the objects themselves.

Key points

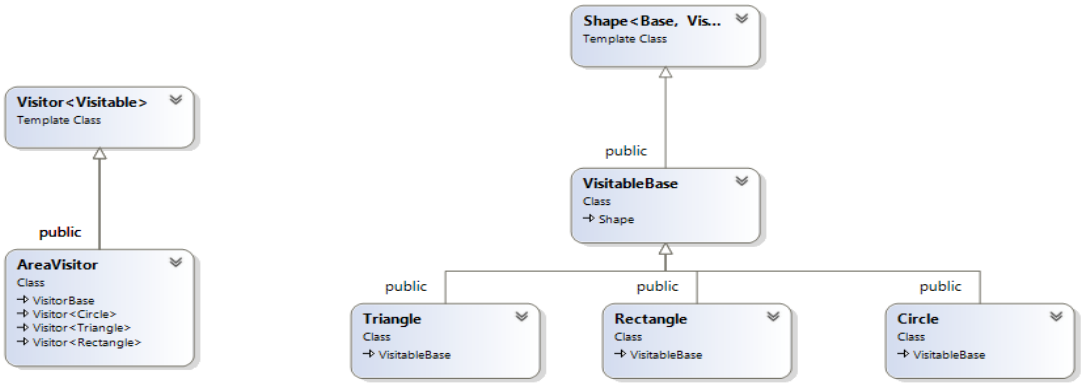
1. We have hierarchy of shapes (Circle, Rectangle and Triangle)
2. We would use Visitor base class with pure virtual method called Visit() which is implemented by the AreaVisitor and is called for every Shape object
3. Visitable base classes, providing Accept() methods, from which shape hierarchy (Triangle, Rectangle and Circle) are derived and made to accept external visitor

I have implemented 3 Variations of Visitors (with reference from Hands on Patterns)

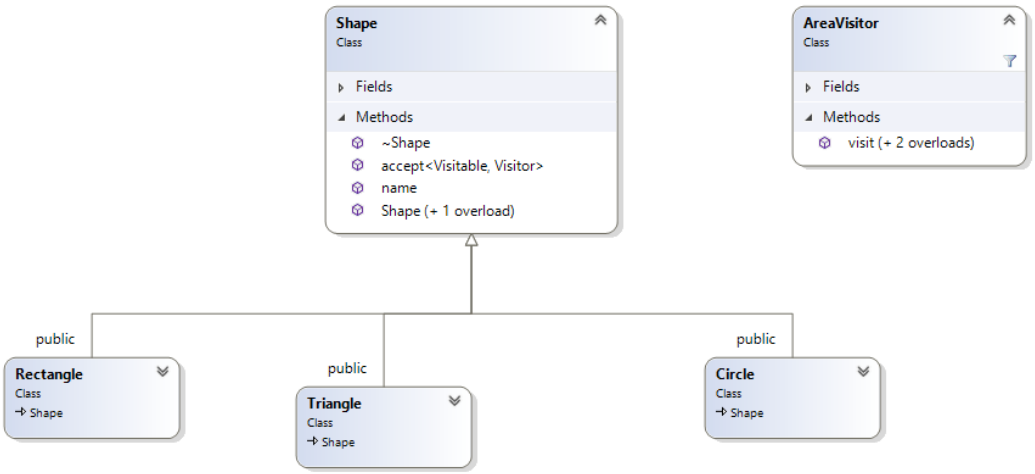
1. **RuntimeVisitorCyclic** –This is classic Visitor pattern implementation, which implements the double dispatch at run-time and he program selects which code to run based on two factors, the types of the visitor and the visitable objects. This implementation **key disadvantage** is that it puts cyclic dependency between Visitor and Visitable classes. Whenever there is new class added to hierarchy, the source code containing the visited hierarchy need to be recompiled.



2. **RuntimeVisitorAcyclic** – This variation of Acyclic Visitor successfully resolves the cyclic dependency issue between Visitor and Visitable class hierarchy. However, it needs to use dynamic cast, which is expensive in terms of performance.




3. **ComplieTimeVisitorSimple** – As opposed to classic Visitor pattern, using double dispatch at run-time—this implementation uses compile time C++ language feature to provide the facility making it much faster at run time and does not have cyclic dependencies issue.




## VISITOR PERFORMANCE

There is few test routines with every visitor class which Acyclic visitor takes 4-5 times more time as compared to cyclic and Classic Visitor due to dynamic cast.

## TESTER LIB

1. **TestRPS:** Boost based basic test cases for RPS (mainly game strategy)
2. **TestVisitor:** Boost based basic test cases for Visitor (testing basic working.)

## DIRECTORY STRUCTURE

a) **scripts** - build scripts for windows and linux

b) **src** – source code for assignment

   a. GameMgr

   b. RPS

   c. main

   d. tester

c) Readme.pdf

d) **vsbuild/build** – generated For build files. Following directories should be created bin, lib,

e) CMakeLists.txt - CMake file for project

## BUILD INSTRUCTIONS:

Please note build has dependency on cmake for make file generation.

**Configurations :** NA

**Build Steps**

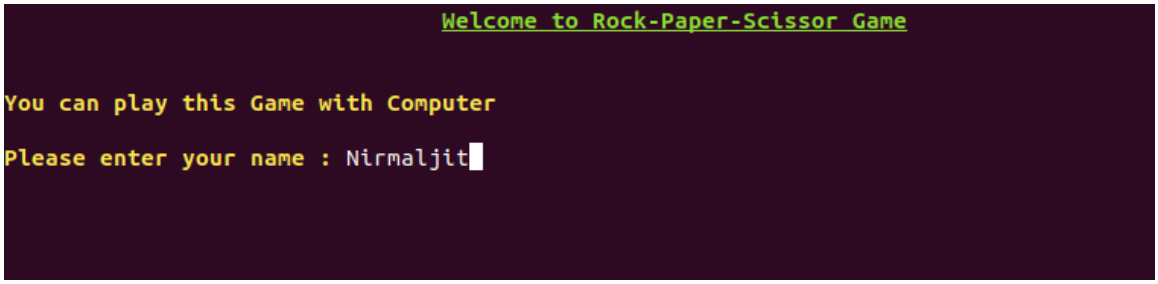
1. unzip Assignment
2. cd scripts
3. run vsbuild.bat or unix_build.sh depending on platform

## RPS GAME SCREEN SHOTS (COMMAND LINE)

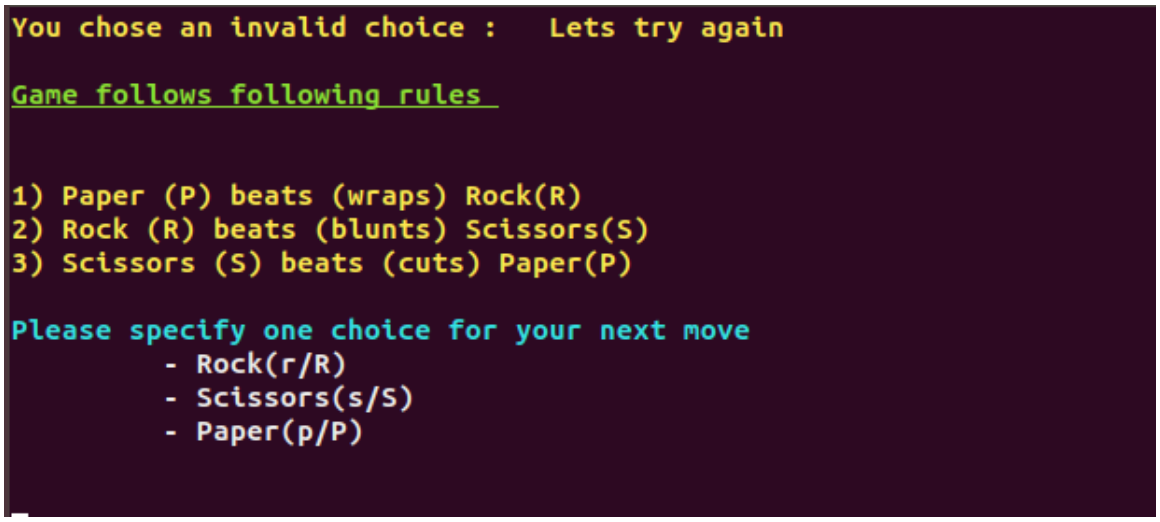
To run the program go to build/bin directory.

./main

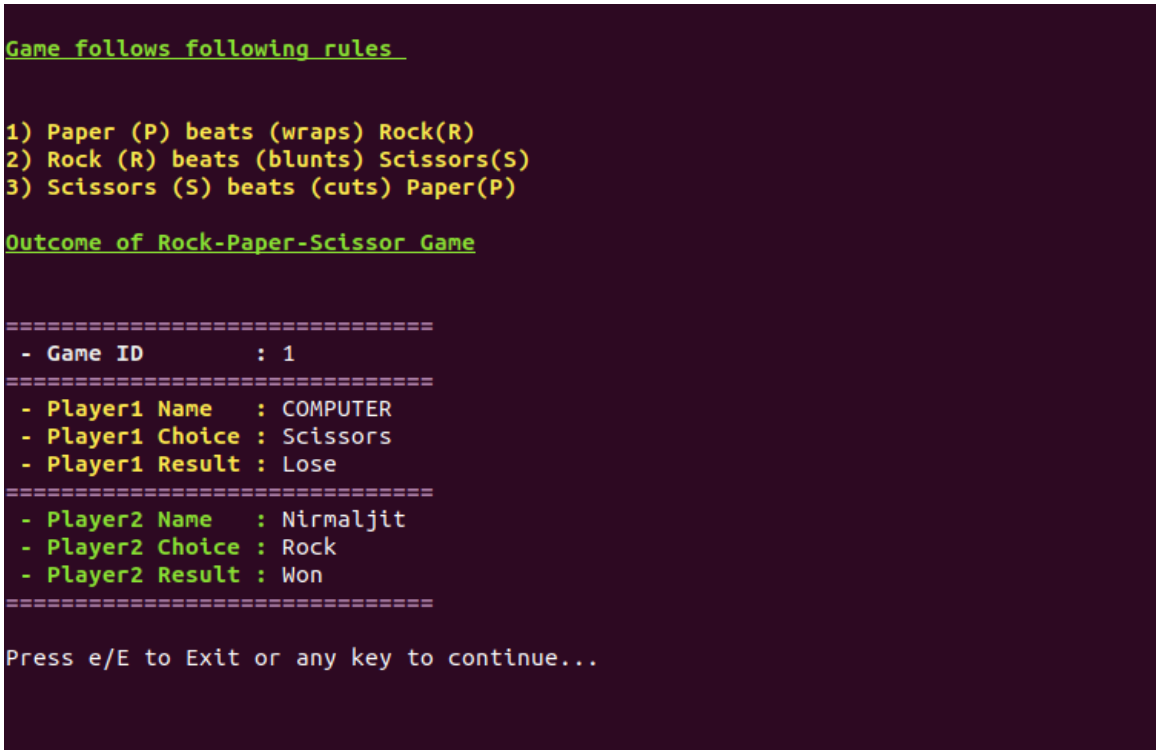
1. Welcome Screen – prompt for name




2. Prompt for next move choice



3. Result Outcome of game




4. Summary report of overall runs