Initially I went and implemented the Auditor class with a method update. This class was my Observer. On the initialization of the Participants auditor instance were added.

From the template method I was calling the auditor instance when a participant/player has made a successful attempt, by passing an instance of the player to the Auditor :

class Auditor

```
 def update(player)
   puts("The Auditor has been updated that #{player.player_name} made a successful attempt")
 end
end
```

This way Auditor was may Observer and the method call was invoked every time something changes.

I thought that I can simplify the solution and make it more generic. Next step was to use the code block as observers.

The advantage of using the Proc as observers is that we are no longer needed Auditor class. This way we can add multiple observers interesting in different changes. On initialization of Participant I called super() constructor with parentheses for an empty argument list. I implemented the module ObservableSubject as follows:

```ruby
module ObservableSubject

    def initialize
      @observers=[]
    end

    def add_observer(&observer)
      @observers.push(observer)
    end

    def delete_observer(observer)
      puts("#{observer.player_name} has been deleted!")
      @observers.delete(observer)
    end

    def notify_observers
      @observers.each do |observer|
        observer.call(self)
      end
    end

end
```

It was Mixin with the Participant class. So in my design I had four Strategies for playing the guessing game, Random, SmartRandom, Binary and Linear. This way I was enabled to add observers for all of them, interesting in different thing. For example I had an observer telling me how long took to SmartRandom and Random to guess the number per run.

```
smart_random_player = SmartRandom.new(1,NUM_OF_RUNS, oracle,
max_num_attempts:NUM_OF_RUNS*5)
smart_random_player.add_observer do |player|
  puts("#{player.player_name} finished the game and took him
#{player.num_attempts} to guess the number!")
end
```

In my derived class SmartRandom and Random I added simply notify observers in one method:

```
def do_play(lower, num, upper)
  while @oracle.is_this_the_number?(num)!=:correct && (@num_attempts <=
@max_num_attempts) do
    num = init_limits(lower, upper)
    @num_attempts+=1
  end
  notify_observers - TELLING ME WHEN HE IS DONE AND HOW MANY ATTEMPTS TOOK HIM
end
```

For the Binary player strategy I was listening when he made a successful attempt.

```
binary_player = Binary.new(1,NUM_OF_RUNS, oracle, max_num_attempts:NUM_OF_RUNS*5)
binary_player.add_observer do |player|
  puts("#{player.player_name} made a successful attempt and took him
#{player.num_attempts}!")
end
```

In the main class after the game is finished I removed each observer. To do this I mixin the ***ObservableSubject*** in my PlayGame interface . Such a way I can call the delete method –

```
players.each { |player|
  total_num_attempts = 0
  total_num_failures = 0
  1.upto(NUM_OF_RUNS) do |i|
    oracle.secret_number = i
    player.reset
    if player.play==:success
      total_num_attempts += player.num_attempts
    else
      total_num_failures += 1
    end
  end
  player.delete_observer(player)
}
```

```ruby
require_relative 'observable_subject.rb'

class PlayGame

  include ObservableSubject

  def initialize(player)
    super()
    @player = player
  end

  def play
    @player.play
  end

  def reset
    @player.reset
  end

  def num_attempts
    @player.num_attempts
  end

  def player_name
    @player.player_name
  end
end
```