Programación Orientada a Objetos – 1er semestre 2011

Trabajo Obligatorio

En el año 1997, el entonces campeón mundial de ajedrez, Gary Kasparov jugó 6 partidas contra la súper computadora de IBM: Deep Blue. Luego de ganar la primera partida y empatar 3, Kasparov perdió dos partidas seguidas y se convirtió en el primer campeón mundial de ajedrez en perder contra una computadora.

Ahora, luego de haber llegado a un punto avanzado en el curso de programación orientada a objetos, deberías ser capaz de desarrollar un software que pueda replicar esta hazaña.

Parte 1 – Diagrama de clases

Identifica todos los objetos básicos necesarios para esta aplicación. Por ejemplo, el tablero* y las piezas. Analiza el diseño de tu aplicación y realiza un diagrama de clases que represente la misma. Este diagrama debe ser entregado junto con tu obligatorio.

**Nota: Un tablero de ajedrez contiene 64 espacios. 32 Blancos y 32 Negros.*

Parte 2 – Implementar el Juego

Desarrolla, utilizando la librería proporcionada, el juego de ajedrez. Para esto deberás respetar y utilizar todas las interfaces que te proveemos. El juego se dará entre dos usuarios y no contra la computadora.

En definitiva, deberán comenzar el juego, manejar las diferentes piezas del juego y finalizar el juego. No deberán preocuparse de las reglas internas del ajedrez, sino de los movimientos de las piezas en sí mismas, considerando los movimientos posibles para cada una, según las ubicaciones del resto de las piezas en el tablero. Deben controlar que no se hagan movimientos inválidos para la partida.

Opcionalmente, pueden idear la solución para que un usuario juegue contra la computadora. Si deciden hacerlo, los movimientos de la computadora sólo validarán sus movimientos posibles y tomarán uno al azar. No es afán de este obligatorio tener un computador inteligente, que analice la mejor jugada.

La librería provista (Lib.rar) contiene los siguientes elementos:

Interfaces:

- *IGui*

```
public interface IGui
{
    /// <summary>
    /// Agrega un listener al IGui. Este listener será llamado
       cada vez que se hace click sobre el menú
    /// </summary>
    /// <param name="listener">Listener que se ejecutará al
       seleccionar una opción del menú</param>
    void AddMenuListener(IMenuListener listener);
    /// <summary>
    /// Agrega un listener al Gui. Este listener será llamado cada
       vez que se haga click sobre una celda del IGui
    /// </summary>
```

```
        /// <param name="listener">Listener que se ejecutará al
            seleccionar una celda del tablero</param>
        void AddCellSelectListener(ICellSelectListener listener);
        /// <summary>
        /// Redibuja el IGui
        /// </summary>
        void Draw();
    }
```

- *IBoard*

```
    public interface IBoard
    {
        /// <summary>
        /// Obtiene la celda indicada del tablero
        /// </summary>
        /// <param name="x">coordenada X de la celda deseada</param>
        /// <param name="y">coordenada Y de la celda deseada</param>
        /// <returns>Celda del tablero</returns>
        ICell GetCell(int x, int y);
        /// <summary>
        /// Obtiene el ancho del tablero
        /// </summary>
        int Width { get; }
        /// <summary>
        /// Obtiene el alto del tablero
        /// </summary>
        int Height { get; }
    }
```

- *ICell*

```
    public interface ICell
    {
        /// <summary>
        /// Agrega una pieza a la celda
        /// </summary>
        /// <param name="piece">Pieza a agregar</param>
        void AddPiece(IPiece piece);
        /// <summary>
        /// Quita la pieza de la celda
        /// </summary>
        void RemovePiece();
        /// <summary>
        /// Obtiene el color de la celda
        /// </summary>
        /// <returns>Color de la celda</returns>
        Color GetColor();
        /// <summary>
        /// Verifica si la celda contiene una pieza
        /// </summary>
        /// <returns>Devuelve true si la celda contiene una
            pieza</returns>
        Boolean HasPiece();
        /// <summary>
        /// Obtiene la pieza en la celda
        /// </summary>
        /// <returns>Pieza</returns>
        IPiece GetPiece();
    }
```

- *IPiece*

```
public interface IPiece
  {
      /// <summary>
      /// Indica si la posición nueva de la pieza es válida
      /// </summary>
      /// <param name="newPos">Posición deseada para la
          pieza</param>
      /// <param name="board">El tablero donde se encuentra la
          pieza</param>
      /// <returns>Devuelve true si el movimiento es
          válido</returns>
      bool IsValidMove(Coordenada newPos, IBoard board);
      /// <summary>
      /// Devuelve una lista de posibles coordenadas a donde la
      /// pieza puede moverse en base a la posición actual
      /// </summary>
      /// <param name="board">El tablero donde se encuentra la
          pieza</param>
      /// <returns>lista de Coordenadas posibles</returns>
      List<Coordenada> ValidMoves(IBoard board);
      /// <summary>
      /// Mueve la pieza de lugar
      /// </summary>
      /// <param name="newPos">Nueva posición de la pieza</param>
      void Move(Coordenada newPos);
      /// <summary>
      /// Obtiene la figura que representa la pieza
      /// </summary>
      /// <returns>Figura que representa la pieza</returns>
      IShape GetShape();
}
```

- *IShape*

```
public interface IShape
  {
      /// <summary>
      /// Debe devolver un Path que representa la forma de la pieza
      /// El path debe estar contenido dentro del recuadro formado
          por los siguientes puntos:
      /// (0,0);(0,100);(100,100);(100,0)
      /// </summary>
      /// <returns>GraphicsPath que representa la figura</returns>
      GraphicsPath GetShape();
      /// <summary>
      /// Devuelve el color de de la figura
      /// </summary>
      /// <returns></returns>
      Color GetShapeColor();
  }
```

- *Listener: ICellSelectListener*

```
 public interface ICellSelectListener
  {
      /// <summary>
      /// Método que se llama cuando se desea invocar el listener
      /// </summary>
      /// <param name="selectedCell">Coordenada de la celda
```

```
            seleccionada</param>
        void Listen(Coordenada selectedCell);
    }
```

- *Listener: IMenuListener*

```
public interface IMenuListener
{
    /// <summary>
    /// Método que se llama cuando se desea invocar el listener
    /// </summary>
    /// <param name="menuText">Nombre de la opción del menú que
        fue seleccionada</param>
    void Listen(string menuText);
}
```

Clases:

- *Color:* Esta clase permite representar hasta 16 millones de colores, en baes a sus componentes rojo (R), verde (G) y azul (B). La intensidad de cada componente puede variar de 0 (sin color) hasta 255. Por ejemplo, un color creado a partir de R=255, G=255 y B=0 generara un color amarillo, mientras que R=0, G=0 y B=255 generara uno puramente azul.
- *Coordenadas*: Representa una coordenada con sus valores de x e y, sus getters y setters.

Forms:

- *Gui:*
    - Implementa la interface IGui
    - Hereda de Form.
    - Se compone con dos listas de listeners, una que corresponde a los listeners del menú, y otra que corresponde a los listeners de las celdas.
    - Maneja las operaciones del menú, desencadenando acciones a través de los listeners.

* Nota: El código que aparece en ésta letra se incluye a los efectos de comprender la letra del obligatorio, pero deberán manejar el código directamente en la librería que les proveemos, cuya versión es la definitiva.

Parte 3 – Detener y Re-anudar el Juego

Utilizando las librerías proporcionadas, diseña alguna forma de almacenar y cargar una partida de ajedrez para poder continuarla en otro momento. Es decir, un jugador, podría pausar el juego y retomarlo luego en el lugar donde lo dejó. Para esto deberás trabajar en dos aspectos:

- Almacenar el juego en un archivo que luego puedas leer, a la hora de detener el juego. Esto se desencadena al darle en el menú la opción de *Save* del GUI.
- Leer el juego que has guardado para reanudarlo. Para esto utilizarás el downloader, y lo que has aprendido en los laboratorios anteriores. Esto se ejecuta, cuando desde el menú se selecciona *Open* del GUI.

Downloader:

```
public class Downloader
{
    private string url;
    /// <summary>
    /// La ubicación de la cual descargar
    /// </summary>
    public string Url { get { return url; } set { url = value; } }

    /// <summary>
    /// Crea una nueva instancia asignando la ubicación de la cual
        descargar
    /// </summary>
    /// <param name="url"></param>
    public Downloader(string url)
    {
        this.url = url;
    }

    /// <summary>
    /// Descarga contenido de la ubicación de la cual descargar
    /// </summary>
    /// <returns>Retorna el contenido descargado</returns>
    public string Download()
    {
        // Creamos una nueva solicitud para el recurso
        especificado por la URL recibida
        WebRequest request = WebRequest.Create(url);
        // Asignamos las credenciales predeterminadas por si el
        servidor las pide
        request.Credentials = CredentialCache.DefaultCredentials;
        // Obtenemos la respuesta
        WebResponse response = request.GetResponse();
        // Obtenemos la stream con el contenido retornado por el
        servidor
        Stream stream = response.GetResponseStream();
        // Abrimos la stream con un lector para accederla más
        fácilmente
        StreamReader reader = new StreamReader(stream);
        // Leemos el contenido
        string result = reader.ReadToEnd();
        // Limpiamos cerrando lo que abrimos
        reader.Close();
        stream.Close();
        response.Close();
        return result;
    }

}
```

Parte 4 – Otro juego

Ya que el ajedrez es un juego que consume mucho tiempo y no todo el mundo sabe jugar, adapta tu programa para que además puedas jugar a otro juego de tablero a elección. Por ejemplo las damas o al shogi (variación japonesa del ajedrez). Es importante que sepas y puedas re-utilizar gran parte del código que tú mismo desarrollaste. Para esto deberás agregar una forma de seleccionar a que juego jugar.