Estándar del juego
Adivina, ¿Qué?

Nombre del curso: Tecnologías para la Construcción de Software
Nombre del profesor: Pérez Arriaga Juan Carlos
Fecha: 16/12/2021
Alumnos:

- Valdes Iglesia Karina
- Vargas Segura Mariana Yazmin

Tabla de contenido

1. Introducción

En el presente documento se describirán las convenciones permitidas para el desarrollo del juego Adivina, ¿Qué? Abarcará las cuestiones de nombres permitidos en todos los componentes del sistema, estilo de código y buenas prácticas. La escritura del estándar se encuentra enfocado para los integrantes del equipo de diseño y construcción del juego Adivina, ¿Qué?; después de llegar al acuerdo de las reglas que deberán ser acatadas en el transcurso del proyecto.

1.1. Propósito y alcance

Este documento describe las reglas, acuerdos y estándares de codificación que se seguirán para llevar a cabo el proyecto del juego Adivina, ¿Qué? con el propósito de desarrollar código de calidad y mejorar la legibilidad. Este documento está escrito específicamente para el proyecto mencionado anteriormente.

1.2. Idioma

El nombre tanto de las variables, clases funciones, paquetes y constantes de nuestro proyecto serán descritos en inglés, esto con la finalidad de adaptarnos al idioma universal para la programación. Considerando que todo programador que quiera implementar mejoras en un futuro manipule dicho idioma. Sin embargo, las GUI serán escritas en español e inglés , debido a que son los idiomas acordados con el maestro de la experiencia educativa.

1.2.1. Internacionalización

En este aspecto los apartados que se considerarán a poseer los idiomas de inglés y español serán todos los componentes gráficos de la GUI, los mensajes de alerta , mientras que las cartas del memorama sólo serán consideradas en el idioma español.

2. Convenciones de nombres

En el siguiente apartado se describen las reglas que se aplicarán en el nombramiento de variables, clases, paquetes, funciones y constantes. Se divide en las secciones de reglas comunes, reglas específicas y los prefijos permitidos en el proyecto.

2.1. Reglas comunes.

El apartado de reglas comunes se refiere a las convenciones que se aplicarán a todas las secciones mencionadas a continuación sin importar si son variables, clases, paquetes, etc.

R-01 Para la declaración de nombres largos, se omitirán las preposiciones.

Ejemplo incorrecto:

void takeOutOfArrayTheFirst(int [] arrayToChange);

Ejemplo correcto:

void TakeOutArrayFirst(int [] arrayChange);

R-02 No se permite el uso de caracteres especiales en la declaración de un nombre.

Ejemplo incorrecto:

int pair_number;

Ejemplo correcto:

int pairNumber;

R-03: Nombre de acuerdo con su propósito.

Ejemplo incorrecto:

int pn;

Ejemplo correcto:

int pairNumber;

2.2. Reglas específicas.

Se exponen los acuerdos y los estilos de escritura que se implementarán a los nombres de cada sección de manera particular.

2.2.1. Clases

R-01: Se usará la notación UpperCamelCase.

Ejemplo incorrecto:

```
public class car{ }
```

Ejemplo correcto:

```
public class Car{ }
```

R-02: Atributos de una clase lowelCamelCase y métodos de una clase deben usar UpperCamelCase.

Ejemplo incorrecto:

```
public class Person{
  private int Age;

  public int GetAge(){
        return Age;
  }


}
```

Ejemplo correcto:

```
public class Person{
  private int age;

  public int GetAge(){
        return age;
  }


}
```

R-03: Los métodos de la clase deben ser nombrados con un verbo en infinitivo.

Ejemplo incorrecto:

void OrderedArray(int [] array);

Ejemplo correcto:

void Sort(int [] arreglo);

R-04: Se deben nombrar con un sustantivo y en singular.

Ejemplo incorrecto:

public class RedAirplanes{ }

Ejemplo correcto:

public class Airplane{ }

2.2.2. Paquetes

R1: Declaración de paquetes todo en mayúsculas

Ejemplo incorrecto:

package menu;

Ejemplo correcto:

package Menu;

2.2.3. Funciones

R1: Se usará la notación UpperCamelCase.

Ejemplo incorrecto:

float calculateAverage(int[] averageArray);

Ejemplo correcto:

float CalculateAverage(int[] averageArray);

2.2.4. Variables

R1: Se usará la notación lowelCamelCase.

Ejemplo incorrecto:

float FinalAverage;

Ejemplo correcto:

float finalAverage;

2.2.5. Constantes

R1: Declaración de constantes todo en mayúsculas.

Ejemplo incorrecto:

int Pi=3.1416;

Ejemplo correcto:

int PI= 3.1416;

R2: Las constantes de 2 o más palabras son separadas por un guion bajo.

Ejemplo incorrecto:

int POSTALCODE = 1012;

Ejemplo correcto:

int POSTAL_CODE = 10;

2.2.6. Propiedades

R1: Se usará la notación UpperCamelCase.

Ejemplo incorrecto:

bool nextTurn{set; get}

Ejemplo correcto:

bool NextTurn{set; get}

2.3. Prefijos permitidos

En la tabla siguiente, se explican los prefijos a utilizar en el proyecto al igual que los casos donde se permitirá el uso de estos.

Prefijo	Significa
bt	Button
lb	Label
tf	TextField
pn	Panel
ta	TextArea
cbx	CheckBox
rb	RadioButton
lt	List
cb	ComboBox
mb	MenuBar
tb	Table
ex	Excepciones
lv	ListView
dp	DatePicker
I	Interface
tv	TableView
tc	TableColumn
sp	ScrollPane
pf	PasswordField
tg	ToggleGroup
lbx	ListBox
pb	Passwordbox
lbi	ListBoxItem

3. Estilo código

En la siguiente sección se especifican las reglas de estilo de código que serán utilizadas durante la etapa de desarrollo del sistema.

3.1. Espaciado

Se describen las convenciones que se utilizarán para ocupar el espaciado dentro del código, con el propósito de proporcionar una mejor legibilidad.

R1: Después del cierre de una función, se deberá dejar una línea en blanco.

Ejemplo incorrecto:

```
public class Person{
    private int age;

        public int GetAge(){
        return age;
                }
        public void SetAge(int age){
                this.age = age;
        }
}
```

Ejemplo correcto:

```
public class Person{
        private int age;

        public int GetAge(){
        return age;
        }

        public void SetAge(int age){
                this.age = age;
        }
}
```

R2: Al finalizar de declarar los atributos de una clase, se deberá dejar una línea en blanco antes de declarar los métodos.

Ejemplo incorrecto:

```
public class Person{
    private int age;
    public int GetAge(){
    return age;
    }

    public void SetAge(int age){
        this.age = age;
    }
}
```

Ejemplo correcto:

```
public class Person{
    private int age;

    public int GetAge(){
    return age;
    }

    public void SetAge(int age){
        this.age = age;
    }
}
```

3.2. Comentarios

Se especifican los acuerdos con respecto a la escritura de código, al igual que los casos donde se utilizarán.

3.2.1. Mono línea

R1: No ha de superar los 45 caracteres.

Ejemplo incorrecto:

// Hello! There is my function “isPair” , the fuction check if a number is pair and return true.

Ejemplo correcto:

// Function by: < name >

3.2.2. Multilínea

R1: Se usarán para especificar si falta la implementación de alguna clase o funciones.

Ejemplo incorrecto:

```
/* Hello! There is my function “isPair”
The fuction check if a number is pair and return true. */
boolean isPair(int numberEvaluate);
```

Ejemplo correcto:

```
/* Lack  implement behavior of fuction
Just I wrote declaration to function  */
boolean isPair(int numberEvaluate);
```

4. Buenas prácticas

En esta sección se enfoca a las convenciones relacionadas con la calidad del producto, al igual que recomendaciones para seguir el estándar.

4.1. Manejo de errores y excepciones

Se describen las determinaciones que se seguirán en las descripciones de los errores que puedan surgir a raíz de cada tipo de excepción a la hora de la ejecución.

R-01: Especificar el error generado por la excepción.

Ejemplo incorrecto:

```
try {
        int numberJoined;
        numberJoined= Int32.Parse (Console.readLine());
        Console.WriteLine("Your number is:  " + numberJoined);
} catch (Exception exception) {
        Console.WriteLine("ERROR");
}
```

Ejemplo correcto:

```
try {
        int numberJoined;
        numberJoined= Int32.Parse (Console.readLine());
        Console.WriteLine("Your number is: " + numberJoined);
} catch (NumberFormatException exNumeroInvalido) {
        Console.WriteLine("Invalid number");
} finally {
        ReadNumber();
}
```

R-02: Usar el bloque *Finally* en las excepciones donde no se necesite interrumpir la ejecución de golpe.

Ejemplo incorrecto:

```
try {
        int numberJoined;
        numberJoined=Integer.parseInt(Console.readLine());
        Console.WriteLine("Your number is:  " + numberJoined);
} catch (Exception exception) {
        Console.WriteLine("Invalid Number");
}
```

Ejemplo correcto:

```
try {
        int numberJoined;
        numberJoined=Integer.parseInt(Console.readLine());
        Console.WriteLine("Your number is: " + numberJoined);
} catch (NumberFormatException exNumeroInvalido) {
        Console.WriteLine("Invalid number");
} finally {
        // Block of code that runs
        ReadNumber();
}
```

R-03: Todas las excepciones que se presenten en el programa serán guardadas en archivo .log para su futura revisión.

Ejemplo incorrecto:

```
try
    {
      host.Open();
      Console.WriteLine("Server is running");

    }
    catch (AddressAccessDeniedException ex)
    {
      Console.WriteLine(ex.Message);
    }
```

Ejemplo correcto:

```
try
    {
      host.Open();
      Console.WriteLine("Server is running");

    }
    catch (AddressAccessDeniedException ex)
    {
      Logs.Error($"Fallo la conexión ({ ex.Message})");
    }
```

4.2. Recomendaciones

En esta sección se enumeran una serie de recomendaciones consideradas como buenas prácticas que se consideren pertinentes para el desarrollo del proyecto, además de proporcionar ejemplos y contraejemplos para asegurarnos de la comprensión de estas.

REC-01: Procurar las funciones hagan la menor cantidad de tareas posibles

Ejemplo incorrecto:

```
float  CalculateAverageStudent(int[] grades){
float additionSubjects=0;
float totalSubjects=grades.length();
        for(int i=0; i<Subjects; i++){
additionSubjects=additionSubjects+grades[i];
        }
        for(int i=0; i<totalSubjects; indice++){
                System.out.println("Grade: "+grades[i]);
        }
        return additionSubjects/totalSubjects;
}
```

Ejemplo correcto:

```
float CalculateAverage(float additionSubjects, float totalSubjects){
        float average=0;
        average = additionSubjects/totalSubjects;
        return average;
}

float CalculateAddition(int[] grades){
        floatfloat additionSubjects=0;
        int totalSubjects=grades.length();
        for(int i=0; i<Subjects; i++){
        additionSubjects=additionSubjects+grades[i];
        }
        for(int i=0; i<totalSubjects; indice++){
        System.out.println("Grade: "+grades[i]);
        }
        return additionSubjects;

}

void ShowGrades(int[] grades){

                for(int i=0; i<grades.length(); i++){
                System.out.println("Grade: "+grades[i]);
                }

        }
```

REC2: Encapsular los atributos

Ejemplo incorrecto:

```
public class Person{
        public int age;
}
```

Ejemplo correcto:

```
public class Person{
private int age;

public int GetAge(){
return age;
                }


public void SetAge(int age){
   this.age = age;
}
```

5. Programación defensiva

En el presente apartado se muestran reglas relacionadas al tema de programación defensiva, con el fin de brindar mayor seguridad al código, al igual que disminuir los posibles errores tanto en el proyecto como los semánticos a la hora de dar mantenimiento al sistema.

PD- D-01: Minimizar el ámbito de las variables

Ejemplo incorrecto:

```
public class Service : IService
   {
          private string smtpServer = ConfigurationManager.AppSettings["SmtpServer"];
      private  string from = ConfigurationManager.AppSettings["EmailAdmin"];
      private  string smtpServer = ConfigurationManager.AppSettings["SmtpServer"];
      private int port = Convert.ToInt32(ConfigurationManager.AppSettings["Port"]);
      private string passwordAdmin = ConfigurationManager.AppSettings["PasswordAdmin"];
 public string SendMail(string to, string asunto, string body)
      {
         string message = "Error al enviar este correo. Por favor verifique los datos o intente más
tarde.";

         string displayName = "Administrador de Adivina ¿Qué? Memorama";
            MailMessage mail = new MailMessage();
            mail.From = new MailAddress(from, displayName);
            mail.To.Add(to);
            mail.Subject = asunto;
            mail.Body = body;
            message = "Exito";
         return message;
      }
}
```

Ejemplo correcto:

```
public class Service : IService
   {

 public string SendMail(string to, string asunto, string body)
      {
         string message = "Error al enviar este correo. Por favor verifique los datos o intente más
tarde.";
string smtpServer = ConfigurationManager.AppSettings["SmtpServer"];
      string from = ConfigurationManager.AppSettings["EmailAdmin"];
       string smtpServer = ConfigurationManager.AppSettings["SmtpServer"];
        int port = Convert.ToInt32(ConfigurationManager.AppSettings["Port"]);
       string passwordAdmin = ConfigurationManager.AppSettings["PasswordAdmin"];



         string displayName = "Administrador de Adivina ¿Qué? Memorama";
            MailMessage mail = new MailMessage();
            mail.From = new MailAddress(from, displayName);
            mail.To.Add(to);
            mail.Subject = asunto;
            mail.Body = body;
```

PD-02: Minimizar la accesibilidad de clases y sus miembros

Ejemplo incorrecto:

```
  public class CallBack : IGameMgtCallback, IPlayerMgtCallback
   {
{
        var option = Alert.ShowDialogWithResponse(username + " " +
Application.Current.Resources["lbInvitation"].ToString(), Application.Current.Resources["btNo"].ToString(),
Application.Current.Resources["btYes"].ToString());

public bool SendInvitationGame(String username)

        bool value = false;
        if (option == AlertResult.Yes)
        {
           value = true;
        }
        return value;
     }
}
```

Ejemplo correcto:

```
  public class CallBack : IGameMgtCallback, IPlayerMgtCallback
   {
public bool SendInvitationGame(String username)
     {
        var option = Alert.ShowDialogWithResponse(username + " " +
Application.Current.Resources["lbInvitation"].ToString(), Application.Current.Resources["btNo"].ToString(),
Application.Current.Resources["btYes"].ToString());
        bool value = false;
        if (option == AlertResult.Yes)
        {
           value = true;
        }
        return value;
     }
}
```

PD-03: Proveer retroalimentación del valor resultante de un método

Ejemplo incorrecto:

```
public AuthenticationStatus Login(string userName, string password)
    {
        AuthenticationStatus status = AuthenticationStatus.Failed;
        string passwordHashed = ComputeSHA256Hash(password);
        using (var context = new AdivinaQueAppContext())
        {
            var Players = (from account in context.Players
                           where account.userName == userName && account.password == passwordHashed
                           select account).Count();
            if (Players > 0)
            {
                status = AuthenticationStatus.Success;
            }
        }
        return status;
    }
```

Ejemplo correcto:

```
public AuthenticationStatus LoginSuccesful(string userName, string password)
    {
        AuthenticationStatus status = AuthenticationStatus.Failed;
        string passwordHashed = ComputeSHA256Hash(password);
        using (var context = new AdivinaQueAppContext())
        {
            var Players = (from account in context.Players
                           where account.userName == userName && account.password == passwordHashed
                           select account).Count();

            if (Players > 0)
            {

                status = AuthenticationStatus.Success;
            }
        }
        return status;
    }
```

- PD-04: Habilitar verificación de tipos en tiempo de compilación

Ejemplo incorrecto:

```
public List<> GetPlayers(){
     return players;
    }
  public void SetPlayers(List<> players){
        this.players=players;
    }
```

Ejemplo correcto:

```
public List<Player> GetPlayers(){
     return players;
    }
  public void SetPlayers(List<Player> players){
        this.players=players;
    }
```

- PD-05: Diseño de interfaces

Ejemplo incorrecto:

```
public void GetTopics(string username)
{
    throw new NotImplementedException
}
```

Ejemplo correcto

```
public bool GetTopics(string username)

{

  Authentication authentication = new Authentication();

 List<String> topics = authentication.getTopics();

 Users[username].ReceiveTopics(topics);

}
```

- PD-06: Verificación de entradas

Ejemplo incorrecto:

```
        private void RegisterBt_Click(object sender, RoutedEventArgs e){
            register();
        }
```

Ejemplo correcto

```
private void RegisterBt_Click(object sender, RoutedEventArgs e)
    {
        if (tbUsername.Text != "" && Password.Password.ToString() != "" &&
tbName.Text != "")
        {
            if (ValidateData() == DataStatus.Correct)
            {
                Register();
            }
            else
            {
                sendMessage(ValidateData());
            }
        }
        else
        {
            MessageBox.Show("Exist empty fields");
        }
    }
```

6. Documentación de código

<sumary>	Resumen que específica la función de una clase o método
<param>	Describe los parámetros de un método
<returns>	Los retornos que posee el método especificado

7. Referencias

J. (s. f.). Visual Studio documentation. Microsoft Docs. https://docs.microsoft.com/en-us/visualstudio/windows/?view=vs-2022

B. (s. f.-a). Documentos de C#: inicio, tutoriales y referencias. Microsoft Docs. https://docs.microsoft.com/es-es/dotnet/csharp/

G. (2021, 11 diciembre). .NET Standard. Microsoft Docs. https://docs.microsoft.com/en-us/dotnet/standard/net-standard