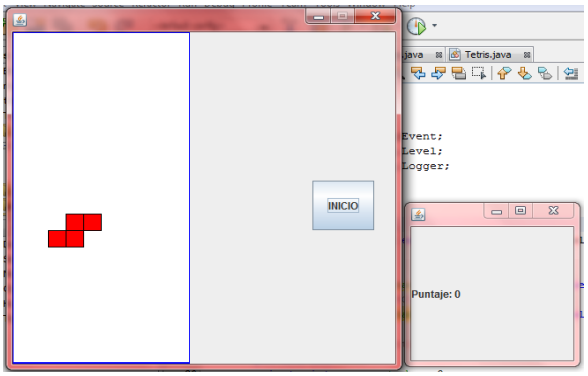
Criterio C: Desarrollo

Técnicas Utilizadas:

1. Interfaz Grafica (Pagina 2)
2. Uso del método paintComponent() y Canvas (Pagina 3)
3. Uso de Hilos y el método run()- Multithreading (Pagina 5)
4. Pensamiento Algorítmico: Movimiento de Piezas (Pagina 6)
5. Clases Anidadas (Pagina 7)

1. INTERFAZ GRAFICA

Tetris es un juego donde el componente visual es de gran importancia debido a que los jugadores deben ver las piezas y el espacio en el cual están. Por esto se decidió utilizar una interfaz grafica debido a que con la programación en consola esto no era posible.



Además la interfaz grafica también proporciona un medio fácil en el cual el usuario puede interactuar con el programa por medio del botón que le da inicio al juego

Este botón es una manera fácil y rápida de empezar el juego y esto prueba ser aun mas importante tomando en cuenta que el proyecto es diseñado para niños.

La interfaz grafica fue creada instanciando el objeto GUI de la clase JFrame y se establecieron los valores de sus atributos. Posteriormente se creó un contenedor donde se situó el JPanel TetrisPanel, que es el lugar donde todo se desarrolla y que fue diseñado utilizando la opción de diseño del IDE Netbeans. Se evidencia mediante esta porción de código:

```
JFrame gui = new JFrame();

gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
gui.setVisible(true);
gui.setSize(440,430);
gui.setResizable(false);

TetrisPanel contenido = new TetrisPanel();


Container principal = gui.getContentPane();

principal.add(contenido);
```

2. Uso del método paintComponent() y la clase Canvas

Mediante la clase Canvas podía realizar los dibujos de los elementos necesarios para tetris, debido a que me permitía formar un lienzo en el cual podía dibujar.

Una interfaz grafica está constituida por diversos elementos visuales que se llaman componentes y la razón por la cual los vemos es que cada uno se dibuja a sí mismo. Pero para dibujar algo nuevo se debe utilizar el método paintComponent() que permite dibujar un componente nuevo sobre la interfaz grafica y en este caso sobre el JPanel. (Eck, 2002)

En primera instancia se utiliza @Override debido a que este método ya está definido previamente y va a ser modificado en el programa. El método recibe un parámetro g del tipo Graphics que representa el contexto grafico que se necesita para poder dibujar sobre el componente.

```
@Override
public void paintComponent(Graphics g) {
```

Se utilizan otros métodos como g.setColor() con el cual se indica el color a utilizar, g.drawRect() que dibuja los bordes del cuadrado y g.fillRect() que lo rellena. El método empieza cubriendo toda el área de blanco para borrar todo lo que estaba anteriormente y después dibuja los bordes.

```
g.setColor(Color.white);
g.fillRect(0, 0, 200, getHeight());

g.setColor(Color.blue);
g.drawRect(0, 0, 200, 400);
g.drawRect(0, 0, 200, 401);
```

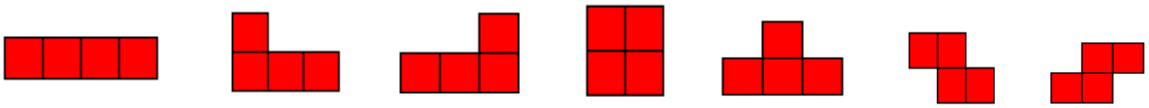
Finalmente el método recorre una matriz de valores booleanos que sirve como referencia. Si el valor de la matriz es verdadero el método dibuja un cuadrado en la coordenada respectiva (j*20,i*20). Se multiplican por 20 para estar en la escala adecuada debido a que cada cuadrado es de 20x20 pixeles. Se utiliza una matriz debido a que permite tratar todos los cuadrados como un grupo y no como objetos individuales.

```
for (int i = 0; i < 20; i++) {
    for (int j = 0; j < 10; j++) {
        if (matrixreference[i][j] == true) {
            g.setColor(Color.RED);
            g.fillRect((j) * 20, i * 20, 20, 20);
            g.setColor(Color.black);
            g.drawRect((j) * 20, i * 20, 20, 20);

        }
    }
}
```

Finalmente se utiliza el método repaint() para indicarle al programa que debe volver a pintar un componente lo cual permite que este se actualice y la animación se cree.

Figuras que se crean mediante esta técnica:



Metodo paintComponent() comentado:

```
//METODO DONDE SE PINTA EL AREA DONDE SE DESARROLLARA EL JUEGO COMO TAL
 @Override
 public void paintComponent(Graphics g) {

     //se borra todo el area
     g.setColor(Color.white);
     g.fillRect(0, 0, 200, getHeight());

     //Se dibuja el borde
     g.setColor(Color.blue);
     g.drawRect(0, 0, 200, 400);
     g.drawRect(0, 0, 200, 401);

     //Se recorre toda la matriz de referencia, si el valor es true se pinta en pantalla
     for (int i = 0; i < 20; i++) {
         for (int j = 0; j < 10; j++) {
             if (matrixreference[i][j] == true) {
                 g.setColor(Color.RED);
                 g.fillRect((j) * 20, i * 20, 20, 20);
                 g.setColor(Color.black);
                 g.drawRect((j) * 20, i * 20, 20, 20);

             }
         }
     }

 }
```

3. Uso de Hilos y del método run()- Multithreading

El uso de un Hilo me proporcionaba los métodos por los cuales podría realizar la animación necesaria para hacer el juego funcionar. Un thread es una instancia del programa y al utilizar Multithreading lo que se estaba buscando era un proceso que pudiese correr de manera paralela e independiente de lo demás procesos, permitiéndole hacer un proceso cíclico e incluso suspenderse por un tiempo. (Oracle, 2013)

Se creó la clase Hilo que extiende de la clase Thread, significando que gracias a la herencia puede utilizar los métodos y atributos de esta clase.

```
private class Hilo extends Thread {
```

Dentro de la clase se utiliza el método run() dentro del cual estará todo lo que el Hilo realizara. Este método también se le debe poner @Override al ser un método nativo a la clase Thread y debe ser modificado. Dentro de este run se ubica un while para poder realizar un proceso cíclico.

```
@Override
public void run() {
```

Otra parte fundamental de el hilo es el método sleep() que permite que el Hilo se suspenda por una porción de tiempo y es lo que permite la animación. En el caso de esta solución el hilo espera 200 milisegundos para que el movimiento sea rápido y fluido.

```
Thread.sleep(200);
```

Finalmente el hilo funciona instanciando en la clase principal TetrisPanel, un objeto de la clase Hilo y utilizando el método start() que hace que el programa empiece a correr todo lo que está dentro del run().

4. Pensamiento Algorítmico: Movimiento de piezas

En este programa el movimiento de las fichas siempre se maneja tomando en cuenta la matriz booleana de referencia y la matriz booleana de movimiento. Cuando una pieza se va a mover el programa empieza identificando las posiciones donde hay un cuadrado y este se está moviendo. Luego revisa si la posición adyacente a la cual se quiere mover está vacía o tiene un cuadrado que también se moverá. Si esto se cumple para todos los cuadrados la figura se mueve en la dirección deseada. Pero si para algún cuadrado hay una pieza estática entonces ningún cuadrado de la ficha se mueve y se convierte también en estático.

Este proceso se vuelve aun más complejo cuando la ficha se rota y se toma en cuenta que ficha es y cuantas veces ha sido rotada:

```
if (tipoPieza == 0 && vecesrotado == 0 && primeravez == 0){
    primeravez++;
    for (int i = 19; i >= 0; i--) {
        for (int j = 0; j < 10; j++) {
            if (matrixreference[i][j] == true && matrixmoviendose[i][j] == true) {
                controtar++;
            }

            if (controtar == 3) {
                System.out.println("" + i + " " + (i - 2));

                if (matrixreference[i - 2][j] == false && matrixreference[i - 1][j] == false && matrixreference[i + 1][j] == false) {
                    matrixreference[i][j - 2] = false;
                    matrixreference[i - 2][j] = true;
                    matrixreference[i][j - 1] = false;
                    matrixreference[i - 1][j] = true;
                    matrixreference[i][j + 1] = false;
                    matrixreference[i + 1][j] = true;

                    vecesrotado++;
                    controtar++;
                    break;
                }

            }
        }

    }
}
```

5. Clases Anidadas

En esta solución se hace uso del concepto de clases anidadas donde se crea una clase dentro de otra clase. En este caso la clase Hilo esta dentro de la clase TetrisPanel.

Esto se hace con el propósito de crear una estructura más lógica, que sea más legible y que tenga una mayor encapsulación. En primera instancia la clase Hilo solo es útil para TetrisPanel, donde se desarrollan todos los procesos, así que es lógico ponerla dentro debido a que no será usada por más clases. Además de esto la encapsulación de la clase aumenta debido a que los atributos privados de la clase externa pueden ser utilizados directamente por la clase interna y además al declarar prívate la clase interna, esta también es protegida.

El anidar clases se hace con el propósito de crear una estructura más lógica, que sea más legible y que tenga una mayor encapsulación. (Oracle, 2011)

Número de Palabras: 1000

Bibliografía

Eck, D. (2002, Julio). *Graphics and Painting.* Retrieved xx, from Introduction to Programming Using Java: http://www.faqs.org/docs/javap/c6/s3.html

Oracle. (2011, Diciembre 12). *Clases Anidadas.* Retrieved xx, from http://devel.no-ip.org/programming/languages/java/tutorial/index.html

Oracle. (2013). *Class Thread.* Retrieved xx, from http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html