UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE
FACULTAD DE INGENIERÍA
CARRERA: INGENIERÍA EN CIENCIAS Y SISTEMAS

PROYECTO 2

MANUAL TÉCNICO

Docente: Ing. Oliver Ernesto Sierra Pac
Curso: Introducción a la Programación y C.
Estudiante: Rudy Alessandro Reyes Oxláj
Registro Académico: 202031213
Fecha: 09/05/2022

MANUAL TÉCNICO

1. DESCOMPOSICIÓN:

1. Tenemos que crear 2 juegos, uno de Damas, y otro de las Torres de Hanoi.
2. Deberá de existir un menú principal que conecte estos dos juegos.
3. Al seleccionar un juego, deberá aparecer un apartado para registrar a los jugadores y así empezar la partida.

Para el juego de Damas:

1. Crear un lógica del juego.
2. Crear la lógica de los movimientos de los peones.
3. Realizar la lógico del movimiento de las reinas.
4. Ver quién fue el ganador, y el perdedor
5. Luego implementar una interfaz gráfica con la lógica del juego.
6. Añadir botones para guardar partida, cargar partida y volver al menú principal.
7. Añadir tiempo, contadores para llevar el registro de la partida.
8. Por ultimo añadir otra ventana para mostrar las estadísticas del juego.
9. En la ventana de estadísticas agregar diferentes botones como los ordenamientos.

Para el juego de las Torres de Hanoi:

1. Crear la lógica del movimiento de los discos.
2. Implementar un método que nos resuelva el juego.
3. Implementar la cantidad de discos que el usuario desea jugar.
4. Crear una interfaz gráfica con la lógica del juego.
5. Añadir botones para guardar partida, cargar partida y volver al menú principal.
6. Añadir los botones para el modo rápido.
7. Añadir los botones de truco y y rendirse.
8. Llevar el control de los movimientos y el tiempo del jugador.
9. Por ultimo añadir otra ventana para mostrar las estadísticas del juego.

2. GENERALIZACIÓN:

1. Crear una ventana de Inicio que conecte a ambos juegos.
2. Crear una ventana para el manejo de usuarios del juego de Damas, para que guarde a los 2 jugadores que empezarán la partida.
3. Crear una ventana para el manejo de usuarios del juego de Torres de Hanoi, para que guarde al jugador que jugará la partida.
4. Crear una ventana que implementará el juego de Damas.
5. Crear una ventana que implementará el juego de Torres de Hanoi.
6. Crear una ventana que maneje todos los reportes del juego de damas.
7. Crear una ventana que maneje todos los reportes de las Torres de Hanoi.
8. Implementar botones de guardar partida, cargar partida, regresar al menú principal, para la ventana del juego de Damas y Torres de Hanoi.

3. ABSTRACCIÓN:

Para el juego de Damas:

1. La ventana de control de usuarios permitirá a los jugadores introducir el nombre del usuario, además habrá un cuadro de texto que muestre todos los jugadores que fueron registrados anteriormente, con el fin de que el jugador pueda usar elegir un usuario ya creado o crear uno nuevo, esta ventana también permitirá que los jugadores elijan el modo de juego, ya sea contra jugador o contra la computadora.

2. Para la ventana que tendrá toda la lógica del juego deberá de tener un menú desplegable en la parte superior que tenga la opción de guarda y cargar partida, además de la opción de regresar al menú principal, también deberá de llevar un cronómetro y un contador de piezas comidas por cada jugador, luego de que termine la partida deberá desplegar las estadísticas del juego.

3. Para la ventana de estadísticas del juego, deberá contener los siguientes reportes:

    Total de partidas jugadas por jugador.
    Total de partidas ganadas por jugador.
    Total de partidas perdidas por jugador.
    Total de movimientos realizados por jugador a lo largo de las partidas.

Promedio de movimientos realizados por jugador a lo largo de las partidas.

Victoria más corta por jugador (partida con la que se requirió la menor cantidad de movimientos para ganarla).

Esta ventana permitirá ordenar los reportes en base a las victorias y derrotas ya sea de forma ascendente o descendente, habrá botones para jugar de nuevo y regresar al menú principal.

Para el juego de las Torres de Hanoi:

1. La ventana de control de usuarios permitirá al jugador introducir el nombre del usuario, además habrá un cuadro de texto que muestre todos los jugadores que fueron registrados anteriormente, con el fin de que el jugador pueda usar elegir un usuario ya creado o crear uno nuevo, esta ventana también permitirá que el jugador elija la cantidad de discos que desea jugar y el modo de juego, ya sea el modo normal o el modo rápido.
2. Para la ventana que tendrá toda la lógica del juego deberá de tener un menú desplegable en la parte superior que tenga la opción de guarda y cargar partida, además de la opción de regresar al menú principal, también deberá de llevar un cronómetro y un contador de movimientos que el usuario va realizando durante la partida, además también deberá de integrar una opción de rendirse.
3. Para el modo normal, el usuario podrá jugar en base a las reglas del juego, podrá elegir hacia donde mover los discos, además habrá implementada una opción de truco que mostrará en pantalla hacia donde tiene que mover el jugador el disco.
4. Para el modo rápido, el usuario no podrá mover los discos de manera manual, solo podrá hacerlo mediante botones, estos serán 3, el primer botón llamado “avanzar” permitirá que los discos se muevan una cierta cantidad de movimientos digitados previamente por el usuario, el botón “retroceder” permitirá regresar cierta cantidad de movimientos los discos y el botón “solución rápida” permitirá resolver el juego de una vez.
5. Para la ventana de estadísticas del juego, deberá contener los siguientes reportes:

    Total de partidas ganadas.

    Total de partidas perdidas.

    Total de partidas abandonadas.

    Promedio de movimientos realizados.

    Promedio de tiempo total.

    Jugador con más partidas ganadas.

Jugador con más partidas perdidas.

Esta ventana permitirá ordenar los reportes en base a las victorias y derrotas ya sea de forma ascendente o descendente, habrá botones para jugar de nuevo y regresar al menú principal.

4. ALGORITMOS:

Lógica del juego de damas:

Clase PiezasTablero:

```
//ATRIBUTOS
int piezasTablero[][];
int rojas = 1, blancas = 2, rojasR = 3, blancasR = 4, vacio = 5;
boolean turno; //True jugador1, false jugador2

método cambiarTurno(){
Si (turno) {
        turno = false;
} de lo contrario {
        turno = true;
    }
}

método verificarFicha(){
boolean seleccionFicha = false;
    Si (turno) {
       if (piezasTablero[x][y] == rojas o piezasTablero[x][y] o rojasR) {
          seleccionFicha = true;
       }
    } de lo contrario {
       Si (piezasTablero[x][y] == blancas o piezasTablero[x][y] == blancasR) {
          seleccionFicha = true;
       }
    }

    retornar seleccionFicha;
}

método verificarRey(){
Para i = 0; hasta i < piezasTablero[0].length; i++) {
        Si (piezasTablero[0][i] == rojas) {
           piezasTablero[0][i] = rojasR;
```

```
        } De lo contrario Si (piezasTablero[7][i] == blancas) {
            piezasTablero[7][i] = blancasR;
        }
    }
}
```

método **moverFicha()**{

```
boolean movimientoCorrecto = false;
    Si (turno) {
        //TURNO DE LAS ROJAS
        Si (piezasTablero[xInicial][yInicial] == rojas) {
            Si ((xInicial - 1) == xFinal y ((yInicial - 1) == yFinal || (yInicial + 1) == yFinal)) {
                Si (piezasTablero[xFinal][yFinal] == vacio) {
                    piezasTablero[xInicial][yInicial] = vacio;
                    piezasTablero[xFinal][yFinal] = rojas;
                    movimientoCorrecto = true;
                }
                 } De lo contrario ((xInicial - 2) == xFinal y ((yInicial - 2) == yFinal || (yInicial + 2) ==
yFinal)) {
                Si ((yInicial - 2) == yFinal) {
                        Si (piezasTablero[xInicial - 1][yInicial - 1] == blancas o piezasTablero[xInicial - 1]
[yInicial - 1] == blancasR) {
                            movimientoCorrecto = movimientoPiezasBR(xInicial, yInicial, xFinal, yFinal, 2,
rojas);
                    }
                }
                De lo contrario Si (yInicial + 2 == yFinal) {
                        Si (piezasTablero[xInicial - 1][yInicial + 1] == blancas o piezasTablero[xInicial - 1]
[yInicial + 1] == blancasR) {
                            movimientoCorrecto = movimientoPiezasBR(xInicial, yInicial, xFinal, yFinal, 2,
rojas);
                    }
                }
            }
        } De lo contrario Si (piezasTablero[xInicial][yInicial] == rojasR) {
              movimientoCorrecto = movimientoReina(xInicial, yInicial, xFinal, yFinal, rojasR, blancas,
blancasR);
        }
    } De lo contraro {
        //TURNO DE LAS BLANCAS
        Si (piezasTablero[xInicial][yInicial] == blancas) {
            Si ((xInicial + 1) == xFinal && ((yInicial - 1) == yFinal || (yInicial + 1) == yFinal)) {
```

```
            Si (piezasTablero[xFinal][yFinal] == vacio) {
               piezasTablero[xInicial][yInicial] = vacio;
               piezasTablero[xFinal][yFinal] = blancas;
               movimientoCorrecto = true;
            }
         } De lo contrario Si ((xInicial + 2) == xFinal && ((yInicial - 2) == yFinal || (yInicial + 2) ==
yFinal)) {
            Si(yInicial-2==yFinal) {
                  Si (piezasTablero[xInicial + 1][yInicial - 1] == rojas || piezasTablero[xInicial + 1]
[yInicial - 1] == rojasR) {
                     movimientoCorrecto = movimientoPiezasBR(xInicial, yInicial, xFinal, yFinal, 1,
blancas);
               }
            }
            De lo contrario SI (yInicial+2 == yFinal) {
                  Si (piezasTablero[xInicial + 1][yInicial + 1] == rojas || piezasTablero[xInicial + 1]
[yInicial + 1] == rojasR) {
                     movimientoCorrecto = movimientoPiezasBR(xInicial, yInicial, xFinal, yFinal, 1,
blancas);
               }
            }
         }
      } De lo contrario Si (piezasTablero[xInicial][yInicial] == blancasR) {
         movimientoCorrecto = movimientoReina(xInicial, yInicial, xFinal, yFinal, blancasR, rojas,
rojasR);
      }
   }
   retornar movimientoCorrecto;
}
```

método **movimientoPiezasBR()**{

```
boolean movimientoCorrecto = false;
   Si (piezasTablero[xFinal][yFinal] == vacio) {
      piezasTablero[xInicial][yInicial] = vacio;
      Si (opcionMetodo == 1) {
         eliminarFichasRojas(xInicial, yInicial, yFinal);
      } De lo cntrario {
         eliminarFichasBlancas(xInicial, yInicial, yFinal);
      }
      piezasTablero[xFinal][yFinal] = colorPieza;
      movimientoCorrecto = true;
   }
```

```
    retornar movimientoCorrecto;
}

método movimientoReina(){
boolean movimientoCorrecto = false;

    Si (xInicial > xFinal) {
      //VA DE ABAJO HACIA ARRIBA
      Para (int i = 1; hasta i < piezasTablero.length; i++) {
            Si ((xInicial - i) == xFinal && ((yInicial - i) == yFinal) || (xInicial - i) == xFinal &&
((yInicial + i) == yFinal)) {
          Si (piezasTablero[xFinal][yFinal] == vacio) {
                eliminarFichasReinasAbajoArriba(xInicial, yInicial, xFinal, yFinal, piezaEnemiga,
piezaReyEnemiga);
            piezasTablero[xInicial][yInicial] = vacio;
            piezasTablero[xFinal][yFinal] = colorPieza;
            movimientoCorrecto = true;
            break;
          }
        }
      }
    } De lo contrario {
      //VA DE ARRIBA HACIA ABAJO
      Para (int i = 1; hasta i < piezasTablero.length; i++) {
            Si ((xInicial + i) == xFinal && ((yInicial - i) == yFinal) || (xInicial + i) == xFinal &&
((yInicial + i) == yFinal)) {
          Si (piezasTablero[xFinal][yFinal] == vacio) {
                eliminarFichasReinasArribaAbajo(xInicial, yInicial, xFinal,yFinal, piezaEnemiga,
piezaReyEnemiga);
            piezasTablero[xInicial][yInicial] = vacio;
            piezasTablero[xFinal][yFinal] = colorPieza;
            movimientoCorrecto = true;
            break;
          }
        }
      }
    }
    retornar movimientoCorrecto;
}
```

método **eliminarFichaReinaAbajoArriba()**{

```
romper:
      Para (int i = 1; hasta i < piezasTablero.length; i++) {
         Si (xInicial - i >= 0 && yInicial - i >= 0) {
            Si ((xInicial - i == xFinal) && (yInicial -i )== yFinal) {
               Para(int j=1;  hastaj<=(yInicial-yFinal); j++) {
                     Si (piezasTablero[xInicial - j][yInicial - j] == enemiga || piezasTablero[xInicial - j]
[yInicial - j] == enemigaR) {
                        piezasTablero[xInicial - j][yInicial - j] = vacio;
                        dibujarTablero.contadorPiezas();
                        break romper;
                     }
                  }
               }
            }
            Si (xInicial - i >= 0 && yInicial + i < 8) {
               Si (xInicial - i == xFinal && (yInicial +i )== yFinal) {
                  Para(int j=1; hastaj<=(yFinal-yInicial); j++) {
                        Si (piezasTablero[xInicial - j][yInicial + j] == enemiga || piezasTablero[xInicial - j]
[yInicial + j] == enemigaR) {
                        piezasTablero[xInicial - j][yInicial + j] = vacio;
                        dibujarTablero.contadorPiezas();
                        break romper;
                     }
                  }
               }
            }
      }
}
```

método **eliminarFichaReinaArribaAbajo()**{

```
romper:
      Para (int i = 1; hasta i < piezasTablero.length; i++) {
         Si (xInicial + i <= 7 && yInicial - i >= 0) {
            Si ((xInicial + i == xFinal) && (yInicial -i )== yFinal) {
               Para(int j=1; hasta j<=(yInicial-yFinal); j++) {
                     Si (piezasTablero[xInicial + j][yInicial - j] == enemiga || piezasTablero[xInicial + j]
[yInicial - j] == enemigaR) {
                        piezasTablero[xInicial + j][yInicial - j] = vacio;
                        dibujarTablero.contadorPiezas();
                        break romper;
                     }
```

```
                }
            }
        }
        Si (xInicial + i <= 7 && yInicial + i < 8) {
            Si ((xInicial + i) == xFinal && (yInicial+i)==yFinal)  {
                Para (int j=1; hasta j<=(yFinal-yInicial); j++) {
                     Si (piezasTablero[xInicial + j][yInicial + j] == enemiga || piezasTablero[xInicial + j]
[yInicial + j] == enemigaR) {
                        piezasTablero[xInicial + j][yInicial + j] = vacio;
                        dibujarTablero.contadorPiezas();
                        break romper;
                    }
                }
            }
        }
    }
}
```

método **llenarTablero()**{

```
Para (int i = 0; hasta i < piezasTablero.length; i++) {
        Para (int j = 0; hasta j < piezasTablero[0].length; j++) {

            Si (j % 2 != 0) {
                piezasTablero[0][j] = blancas;
                piezasTablero[2][j] = blancas;
                piezasTablero[4][j] = vacio;
                piezasTablero[6][j] = rojas;

            } De lo contrario {
                piezasTablero[1][j] = blancas;
                piezasTablero[3][j] = vacio;
                piezasTablero[5][j] = rojas;
                piezasTablero[7][j] = rojas;
            }
        }
    }
}
```

Lógica del juego de Torres de Hanoi:

Clase BloquesTorres:

```
//ATRIBUTOS
 int cantBloques;
 int [][] bloques;
 DibujarTorres dibujarTorres;
 int [][] solucion;
int contador = 0;

método llenarBloques(){
int bloquesMayor = 0;
        Para(int j=0; hasta j<cantBloques; j++){
           bloques[0][j]=8-bloquesMayor;
           bloquesMayor++;

           bloques[1][j] = bloqueVacio;
           bloques[2][j] = bloqueVacio;
        }
}

método verificarGanador(){
   int bloqueMayor = 7;
      boolean ganador = false;
      //TORRE 1 O 2------ 8, 7, 6, 5, 4, 3, 2 ,1
      Si(retornarPosicionBloqueUltimo(1)==cantBloques-1) {
         Para (int i = 0; hasta i < cantBloques; i++) {
            Si (bloques[1][i] > bloqueMayor) {
               ganador = true;
               bloqueMayor--;
            } De lo contrario {
               break;
            }
         }
      }

      bloqueMayor = 7;
      Si(!ganador){
         Si(retornarPosicionBloqueUltimo(2)==cantBloques-1) {
            Para (int i = 0;  hasta i < cantBloques; i++) {
               Si (bloques[2][i] > bloqueMayor) {
                  ganador = true;
```

```
                bloqueMayor--;
            } De lo contrario {
                break;
            }
        }
    }
}
retornar ganador;
}

método activarTruco(){
boolean correcto = false;

    //BUSCAR CUAL ES EL BLOQUE QUE VAMOS A MOVER
    int bloqueMover = retornarBloqueUltimo(torreInicial);
    int posicionBloqueMover = retornarPosicionBloqueUltimo(torreInicial);
    //BUSCAR CUAL ES EL BLOQUE A DONDE VAMOS A MOVERLO
    int bloqueTorre = retornarBloqueUltimo(torreFinal);
    int posicionBloqueMovidoTorre = retornarPosicionBloqueUltimo(torreFinal);

    Si(bloqueTorre==0){
        if(posicionBloqueMovidoTorre+1<8){
            correcto = true;
        }
    }De lo contrario Si(bloqueMover<bloqueTorre){
        Si(posicionBloqueMovidoTorre+1<8){
            correcto = true;
        }
    }
    return correcto;
}

método moverBloque(){
boolean correcto = false;

    //BUSCAR CUAL ES EL BLOQUE QUE VAMOS A MOVER
    int bloqueMover = retornarBloqueUltimo(torreInicial);
    int posicionBloqueMover = retornarPosicionBloqueUltimo(torreInicial);
    //BUSCAR CUAL ES EL BLOQUE A DONDE VAMOS A MOVERLO
    int bloqueTorre = retornarBloqueUltimo(torreFinal);
    int posicionBloqueMovidoTorre = retornarPosicionBloqueUltimo(torreFinal);
```

```
    Si(bloqueTorre==0){
        Si(posicionBloqueMovidoTorre+1<8){
            bloques[torreInicial][posicionBloqueMover] = bloqueVacio;
            bloques[torreFinal][posicionBloqueMovidoTorre] = bloqueMover;
            correcto = true;
        }
    }De lo contrario Si(bloqueMover<bloqueTorre){
        Si(posicionBloqueMovidoTorre+1<8){
            bloques[torreInicial][posicionBloqueMover] = bloqueVacio;
            bloques[torreFinal][posicionBloqueMovidoTorre+1] = bloqueMover;
            correcto = true;
        }
    }
    retornar correcto;
}

método retornarBloqueUltimo(){
int bloqueUltimo = 0;
    Para( int i=(bloques[torreBuscar].length-1); hasta i>=0; i-- ){
        Si(bloques[torreBuscar][i]!=0){
            bloqueUltimo = bloques[torreBuscar][i];
            break;
        }
    }
    retornar bloqueUltimo;
}

método retornarPosicionBloqueUltimo(){
int bloqueUltimo = 0;
    Para( int i=(bloques[torreBuscar].length-1); i>=0; i--){
        Si(bloques[torreBuscar][i]!=0){
            bloqueUltimo = i;
            break;
        }
    }
    retornar bloqueUltimo;
}

método movimientoSolucion(){
Si(cantDiscos==1){
        //TORRE INICIO A LA TORRE DESTINO
```

```
            solucion[contador][0] = torreInicio;
            solucion[contador][1] = torreFinal;
            contador++;

        }De lo contrario{
            movimientoSolucion(cantDiscos-1,torreInicio,torreFinal,torreApoyo);
            solucion[contador][0] = torreInicio;
            solucion[contador][1] = torreFinal;
            contador++;
            movimientoSolucion(cantDiscos-1,torreApoyo,torreInicio,torreFinal);
        }
}
```

método **guardarPartida()**{

```
Para(int i=0; hasta i<bloques.length; i++){
            Para(int j=0; hasta j<bloques[0].length; j++){
                copiaTorres[i][j] = bloques[i][j];
            }
        }
}
```

método **cargarPartida()**{

```
Para(int i=0; hasta i<bloques.length; i++){
            Para(int j=0; hasta j<bloques[0].length; j++){
                bloques[i][j] = copiaTorres[i][j];
            }

            dibujarTorres.moverBloques();
        }
}
```