UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA UTEC

PROGRAMACIÓN II

LABORATORIO 1.01

INFORME DE PROYECTO FINAL

GRUPO 5: LAS VENGADORAS

INTEGRANTES:

Nombre	Código	Coevaluación
Alejandra Soledad Castro Huerta	202010223	100%
Samanta Susana Chang Kuoman	202110074	100%
Delia Katherine Juy Luyo	202110178	100%
Ana Joaquina Mendoza Tejada	202110218	100%
Aixa Ximena Torres Fuertes	202110367	100%

PROFESORA: MARÍA HILDA BERMEJO

2021-2

Lima, Perú

ÍNDICE

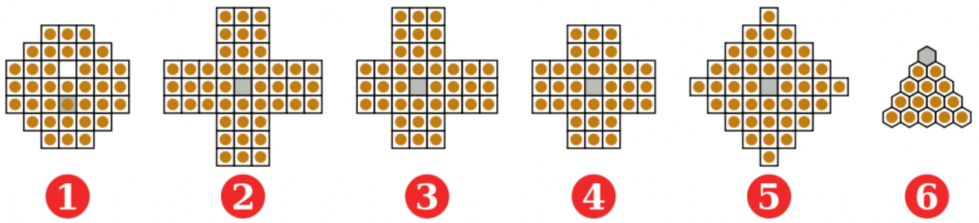
INTRODUCCIÓN Y ANTECEDENTES

Introducción

Diseñar y desarrollar un Programa Orientado a Objetos, que permita reproducir el juego Senku. Los espacios ocupados se representan por O y los espacios vacíos se representan por +. La regla del juego es mover los espacios ocupados hacia los espacios vacíos, pero para moverlo es requisito obligatorio saltar sobre solo un espacio que esté ocupado, de otro modo el espacio ocupado no podrá moverse. Se gana el juego si al final se queda con solo un espacio ocupado y se pierde si no se puede realizar más movimientos y se tiene más de un espacio ocupado, los movimientos solo son verticales u horizontales, pero no diagonales.

```
   1     2     3     4     5     6     7
1                O---- O---- O
                 |     |     |
2                O---- O---- O
                 |     |     |
3  O---- O---- O---- O---- O---- O---- O
   |     |     |     |     |     |     |
4  O---- O---- O---- +---- O---- O---- O
   |     |     |     |     |     |     |
5  O---- O---- O---- O---- O---- O---- O
                 |     |     |
6                O---- O---- O
                 |     |     |
7                O---- O---- O
```

Se solicita realizar al menos 3 de las 6 variantes del juego:



Donde:
1. Estilo francés.
2. Estilo alemán.
3. Estilo asimétrico.
4. Estilo inglés.
5. Estilo diamante.
6. Estilo triangular

Resumen del trabajo

El proyecto desarrollado para la clase del laboratorio de programación orientada a objetos se define como un programa que le permita al usuario poder jugar Senku, un juego de tablero solitario abstracto. El juego consta de un tablero lleno de piezas salvo un hueco, siendo las piezas distribuidas diferente dependiendo del tablero seleccionado. Para poder ganar el juego se deben eliminar todas las piezas del tablero excepto una, para poder eliminar una ficha, se mueve una de las piezas en horizontal o vertical saltando sobre una ficha contigua y acabando en el hueco junto a la pieza saltada que será eliminada. Por lo tanto, sólo es posible eliminar las fichas que están junto a un hueco y de una en una. Para poder desarrollar este programa hemos implementado nuestros conocimientos en la creación de clases, el uso de relaciones entre clases como herencia y la implementación de polimorfismo.

FUNDAMENTO TEÓRICO

DIAGRAMA DE UML

Enlace para acceder:
https://lucid.app/lucidchart/99418ac5-eb6d-4abd-89f6-848aa280f6ce/edit?viewport_loc=-492%2C542%2C3022%2C1198%2C0_0&invitationId=inv_9df794a3-555b-459c-8e25-4e7da44ca342

RELACIONES ENTRE CLASES

¿Qué son las clases en C + +?
Una clase es un conjunto de atributos (características) y métodos (funciones que ejecutan acciones), cada una de estas tiene atributos y métodos públicos o privados, a los cuales se pueden acceder con los métodos de acceso, es decir, getters.

¿Para qué se utilizan las clases?
Las clases se utilizan para agrupar objetos con las mismas características, para poder ejecutar acciones específicas o obtener datos de cada una de ellas para la resolución de algún problema. Al realizar esto, el proceso se optimiza, teniendo como punto de referencia a la programación estructurada.

¿Qué es una relación entre clases?
Es una manera en la que dos o más clases pueden "comunicarse", de manera que datos o acciones específicas contenidas en cada una de ellas sean aprovechadas para optimizar la solución de un determinado problema.

¿Cuáles son los tipos de relaciones entre clases que hay?
Existen 4 tipos principales de relaciones entre clases:

1. **Relaciones de asociación:** Para validar su tipo de relación se utiliza la frase "usa un".
2. **Relaciones de composición:** Existe una fuerte relación de dependencia entre un objeto o más, para validar esta relación se usa la frase "es parte de".
3. **Relaciones de agregación:** Es una relación todo-parte, donde se agrega una parte a un todo, esta relación se valida usando la frase "tiene un".
4. **Relación de herencia:** Es una relación que se valida usando la frase "Es un".

Para este trabajo en particular se usó la relación de herencia.

RELACIONES DE HERENCIA Y POLIMORFISMO

¿Qué es herencia?
La herencia es un tipo de relación entre clases que permite que atributos o métodos que aparecen múltiples veces se puedan agrupar y utilizar en otras clases. Esto se hace con el objetivo de hacer el código lo más conciso posible.

¿Qué es polimorfismo?
El polimorfismo es la habilidad que tienen los objetos de diferentes clases que tienen cierta relación de herencia a responder de manera distinta frente a un mismo mensaje.

- **Polimorfismo por herencia:** El polimorfismo por herencia está relacionada con métodos, específicamente con la herencia de estos, ya sean virtuales puros que responden al mismo mensaje de una manera distinta, es decir, coinciden en el qué y más no en el cómo, o con los métodos que se heredan completamente pues coinciden en el qué y en el cómo.
- **Polimorfismo por sobrecarga de operadores:** El polimorfismo por sobrecarga de operadores se da cuando le puedes dar otra función a un operador.
- **Polimorfismo por uso de templates:** El polimorfismo por uso de templates es necesario cuando se utilizan diferentes tipos de parámetros en los métodos de las clases.

MÉTODOS Y DESARROLLO

ASPECTOS MÁS IMPORTANTES DEL CÓDIGO

Para el desarrollo de este código se utilizaron diferentes archivos en un mismo proyecto, entre estos tenemos tanto archivos ".h" como archivos ".cpp".

Archivos ".h"

En estos archivos se declaran las funciones.

a) Tipos.h
b) Juego.h
c) Funciones.h
d) Estilos.h

En estos archivos se declaran los atributos de las clases y sus métodos correspondientes.

e) **Base_tablero.h:** En este archivo se declara la clase ancestra "Base_tablero" con las variables protegidas piezas y nombres que serán heredadas por las subclases "Base_triangular" y

"Base_cuadrada", los constructores y su respectivo destructor, también se declaran las funciones virtuales puras haciendo de esta una clase abstracta, por lo que no se puede instanciar un objeto de esta clase.

f) **Base_triangular.h:** En este archivo se declara la clase hija "Base_triangular" con las variables privadas alto y ancho, además de los constructores y destructores. Por otro lado, se declaran las funciones virtuales derivadas de la clase padre "Base_tablero", en la cuál con ayuda de polimorfismo y el overriding se usarán adaptándolo a la base triangular del juego.

g) **Base_cuadrada.h:** En este archivo se declara la clase hija "Base_cuadrada" con la variable privada lado, los constructores y su respectivo destructor también con el uso de polimorfismo y overriding se declaran las funciones virtuales derivadas de la clase padre "Base_tablero" para ser adaptadas a la base cuadrada del juego.

Archivos ".cpp"

En estos archivos se desarrollan las funciones declaradas en los archivos ".h".

a) **Juego.cpp:** En este archivo se desarrolla una función de tipo void llamada "juego" en la cuál se validan las posiciones, las piezas y los movimientos a realizar. Además es la función responsable de recibir la opción de movimiento y actuar en base a ello. Al validar los movimientos, se itera toda la matriz de juego, validando si existe la pieza y si puede realizar movimientos (teniendo en cuenta si existen espacios a una pieza de distancia (horizontal o verticalmente) para poder realizar el movimiento) si retorna un verdadero, aún quedan movimientos, y si retorna un falso, no se da el movimiento.

b) **Funciones.cpp:**
   1) **ostream& operator<<(ostream& salida, Base_tablero* tablero):** Sobrecarga el operador "<<" para poder realizar la impresión del tablero rápidamente en la función juego.
   2) **void imprimir_menu():** Realiza la impresión del menú, donde se muestran las opciones para la elección del estilo de juego.
   3) **int seleccionar_opcion_menu():** El jugador ingresa un número entero que debe estar entre el 0 y 7, que representa las opciones del menú, y lo retorna.
   4) **void elegir_posicion(int* x, int* y):** Recibe la posición de la pieza del jugador y los guarda en punteros.
   5) **void imprimir_creditos():** Realiza la impresión de los créditos.

c) **Estilos.cpp:** En este archivo se tienen diferentes funciones para instanciar los tableros de diferentes estilos, desarrollados en una matriz de vectores que contienen valores de tipo char, que se creará según lo establecido en el archivo "main.cpp".

d) **Base_tablero.cpp:** En este archivo se desarrollan los constructores, destructores y métodos de acceso de la clase "Base_tablero" mediante los cuales podemos acceder al nombre del tablero y a la cantidad de sus piezas. Existen funciones virtuales que sirven para validar movimientos y piezas.

e) **Base_cuadrada.cpp:** Deriva de Base_tablero. A partir de base cuadrada, que también es hija de "Base_tablero", se ejecutan 5 modos de juego, del 1 al 5. Tiene 4 comandos de movimiento únicamente: arriba, abajo, derecha e izquierda. Las funciones que se ejecutan sirven para validar la pieza y el movimiento elegido de acuerdo a las funciones existentes en la clase padre.

f) **Base_triangular.cpp:** Deriva de Base_tablero, se acomoda al modo de juego triangular únicamente. La base triangular es una clase hija de base tablero, y a diferencia de la base cuadrada, posee 6 comandos de movimiento: cuatro diagonales (diagonal superior derecho, diagonal superior izquierdo, diagonal inferior derecho, diagonal inferior izquierdo), derecha e izquierda.

   Cabe recalcar que existen dos clases hijas para Base_tablero porque los comandos de movimiento y parámetros son diferentes para el modo triangular y los cinco restantes.

main.cpp

Es donde se implementan todas las funciones para la ejecución del juego.

Se crean las variables flag, de tipo bool, el tablero de juego como un puntero de valor nullptr de la clase Base_tablero y la variable opcion_menu de tipo int.

Seguidamente, se hace uso de la estructura **while**, que permite que el juego se desarrolle, mientras el while sea true, se podrá jugar el juego una y otra vez. Dentro de este while, se realiza lo siguiente:

a) **Selección de modo de juego:** Se le pregunta al jugador que ingrese el número del estilo de juego siguiendo el menú y, utilizando la estructura switch, se define la clase a crear para el tablero, accediendo a Estilos.h, y creando un objeto de la clase Base_cuadrada o Base_triangular. Si el jugador desea dejar de jugar, ingresa "0" y se cambia la variable flag de tipo bool de true a false.
b) **Revisa si la opción ingresada es para dejar de jugar**: Si la variable flag es actualizada a false en el switch anterior, se llama a la función llamar_creditos para imprimir los créditos y termina el programa al romper el while.
c) **Se llama a la función juego**: Si pasa la condición anterior, se llama a la función juego, de Funciones.h, para dar inicio al juego.
d) **Liberación de memoria:** Se elimina el puntero al tablero del juego creado y se le asigna un valor nulo (nullptr).

MANEJO DE ERRORES

Dentro de nuestro código, existen 3 posibles momentos donde el jugador pueda ocasionar errores en el juego: al momento de ingresar el modo de juego, es decir, al escoger el tipo de tablero a jugar, y al momento de ingresar la posición de la pieza a mover y hacia dónde se moverá esta pieza. Se detallará a continuación cómo se manejó cada situación.

Al momento de ingresar modo de juego, la información dada por el jugador se encuentra en un bucle do-while, donde el requisito para salir del bucle es que la información ingresada sea un número mayor igual a 0 y menor a 8, por lo que si el jugador ingresa carácteres no válidos, tendrá que volver a

ingresar la información requerida. Debido a ello, se podrá evitar una mala ejecución del código previniendo que la información que llegue al switch, donde se crea el tablero, sea correcta.

Al momento de ingresar las posiciones de las piezas (x, y), se guardan ambas coordenadas, y se realizan dos validaciones, la primera para saber si dicha pieza existe en el modo de juego elegido, y la segunda para ver si tiene posibilidad de movimiento.

Al momento de ingresar la dirección del movimiento, la información dada está dentro de un while, el cual solo es interrumpido en caso los caracteres sean parte de las teclas de movimiento WASD, o wasd en caso se ingresen en minúscula.

USO DE HERENCIA Y POLIMORFISMO

¿Cómo empleamos herencia en nuestro código?
Se utilizó la herencia con la clase padre “Base_tablero” y sus clases hijas “Base_cuadrada” y “Base_triangular”, teniendo como atributos protegidos al vector de vectores, siendo este el tablero, al número de piezas y al atributo nombre, atributos que todas las clases hijas heredarán.

¿Cómo empleamos polimorfismo en nuestro código?
Lo empleamos mediante el uso de funciones virtuales puras, pues estas coincidían en el qué, mas no en el cómo, por lo tanto, necesitábamos la ayuda del overriding, que nos daría una respuesta diferente ante el mismo mensaje dependiendo al objeto de la clase en cuestión.

De la misma manera, se hace uso de polimorfismo al realizar la sobrecarga del operador “<<”, donde se le asigna la función de imprimir el tablero del juego. La función original de este operador es de las acciones básicas de salida, por lo que al agregarle una función más, hace que dé una respuesta diferente dependiendo del tipo de dato que le siga.

CONCLUSIONES

APRENDIZAJE

Durante el desarrollo de este proyecto pudimos aplicar todos los conocimientos aprendidos en clase, con énfasis en la programación orientada a objetos, sobre todo en el manejo de relaciones de herencia y polimorfismo, lo que nos permitió sellar todo el contenido teórico-práctico.

LIMITACIONES

Las limitaciones que tuvimos durante la realización del proyecto fueron las siguientes: realizar códigos de manera grupal no siempre es sencillo pues los programas colaborativos como Replit no funcionan tan bien como interfaces como Clion y hay algunas funciones que no se pueden emplear. Por otro lado, al resolver errores, como el código es en cierta medida complejo (por su longitud), era difícil hallar el origen.

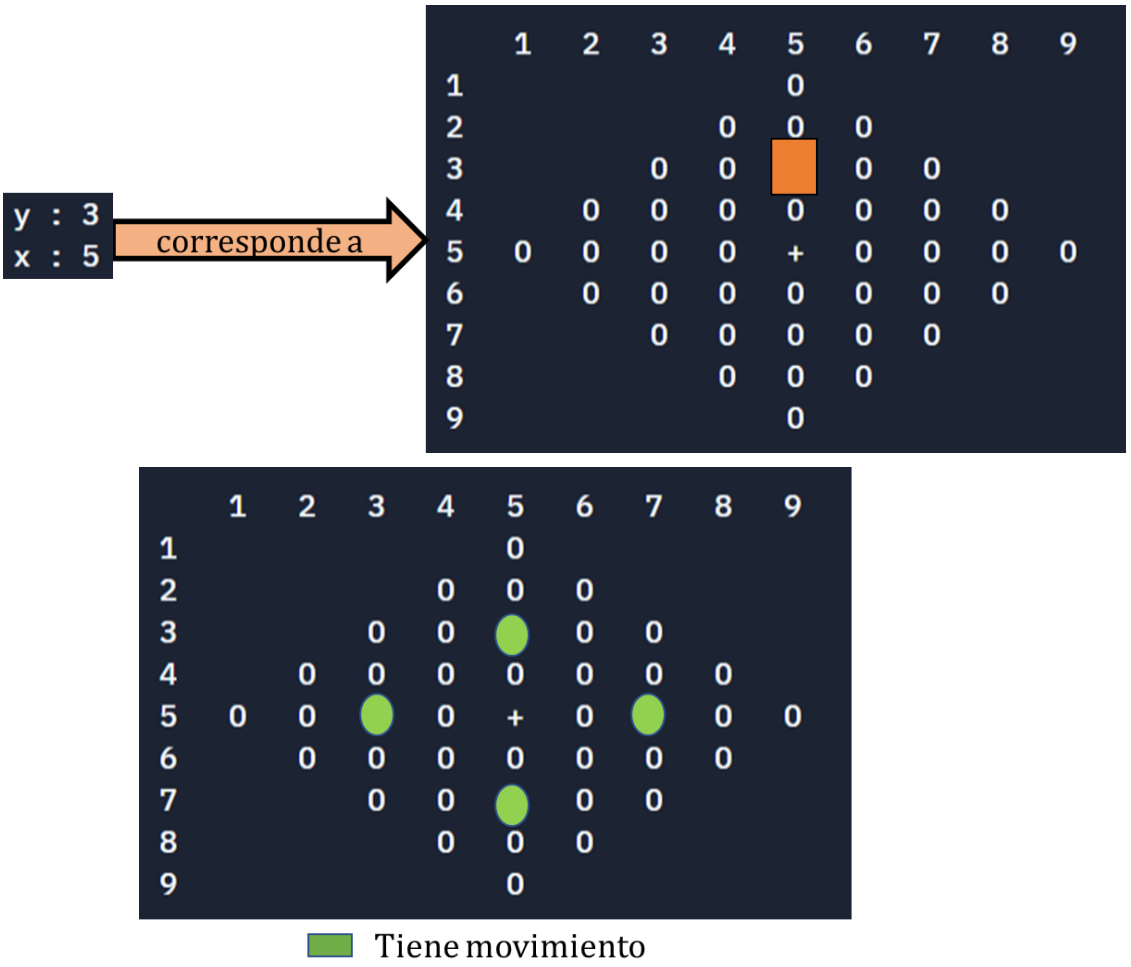
SUGERENCIAS

Para mejorar el proyecto, lo que sugerimos es implementar el proyecto en alguna consola gráfica, para que se vea mejor y a la vez para que sea más fácil para el usuario jugarlo. Por ejemplo, de esta manera se podría visualizar el espacio elegido gráficamente, lo que impide que el jugador se equivoque y tenga que volver a elegir la coordenada.

INSTRUCCIONES DE USO DEL PROGRAMA

1) Elegir el modo de juego desde el menú principal. Al haber incluído todos los estilos de juego, existen 6 modos, por lo que se debe ingresar un número del 1 al 6 (1 - Estilo francés, 2 - Estilo alemán, 3 - Estilo asimétrico, 4 - Estilo inglés, 5 - Estilo diamante, 6 - Estilo diamante). En caso se desee salir del juego, se deberá ingresar el número 0.

2) Posteriormente, se mostrará el tablero según el modo de juego elegido. Lo siguiente será elegir una pieza. Para ello, aparecerá un mensaje "Ingrese ficha a mover (y = vertical, x = horizontal):" y luego aparecerá un "y:" donde colocaremos la componente vertical de la pieza y lo mismo con "x:". Es importante recalcar que se debe elegir una pieza válida (coordenada donde exista una pieza) cercana, máximo a un espacio de la cruz (espacio vacío), de lo contrario aparecerá una notificación que indique lo contrario.
Ejemplo:



3) Al haber elegido una pieza correcta, se podrá mover dicha pieza hacía arriba (W), abajo (S), derecha (D) o izquierda (A). Cuando un movimiento sea realizado, se notificará y se mostrará el espacio vacío nuevo (denotado por una cruz - +) en el tablero.

```
    1  2  3  4  5  6  7  8  9
1                0
2             0  0  0
3          0  0  +  0  0
4       0  0  0  +  0  0  0
5    0  0  0  0  0  0  0  0  0
6       0  0  0  0  0  0  0
7          0  0  0  0  0
8             0  0  0
9                0
```

4) Realizar la mínima cantidad de movimientos para liberar los espacios. Si el usuario se queda sin movimientos, aparecerá una notificación y el juego acabará. Si el usuario gana, también aparecerá una notificación.

```
   1                +
   2             +     +
   3          +     +     +
   4       +     +     +     0
   5    +     +     +     +     +
        1  2  3  4  5  6  7  8  9

FELICITACIONES!!!
HAS GANADO SENKU ESTILO TRIANGULAR!!!
Tablero limpiado
Tablero reiniciado
```

```
   1                +
   2             +     +
   3          0     +     0
   4       +     +     +     +
   5    0     +     +     0     +
        1  2  3  4  5  6  7  8  9

LAMENTABLEMENTE, TE QUEDASTE SIN MOVIMIENTOS
SUERTE PARA LA PROXIMA
Tablero limpiado
Tablero reiniciado
```