Proyecto Final: Implementación del Juego de Ajedrez usando el paradigma de Programación Orientada a Objetos con las herramientas C++ y Qt

1st Alexander Gabriel Luna Choquecota
Escuela Profesional de Ingeniería de Sistemas
Universidad Nacional de San Agustín (UNSA)
Arequipa, Perú
alunach@unsa.edu.pe

2nd Heber Sixto Uraccahua Barrios
Escuela Profesional de Ingeniería de Sistemas
Universidad Nacional de San Agustín (UNSA)
Arequipa, Perú
huraccahua@unsa.edu.pe

3th Efrain Alex Mamani Solorzano
Escuela Profesional de Ingeniería de Sistemas
Universidad Nacional de San Agustín (UNSA)
Arequipa, Perú
emamanis@unsa.edu.pe

4th Oscar Alonso Cuadros Linares
Escuela Profesional de Ingeniería de Sistemas
Universidad Nacional de San Agustín (UNSA)
Arequipa, Perú
ocuadrosl@unsa.edu.pe

***Abstract*—This article details how we implement some of the most relevant features of object-oriented programming, from well-known techniques such as inheritance and polymorphism, to the 'Singleton' design pattern. We use as an object of implementation a well-known and pleasant game, chess.**

***Index Terms*—C++, Qt, singleton, chess, polymorphism, object-oriented programming**

I. Introducción y Motivación

La Programación Orientada a Objetos (POO) involucra distintas características y metodos de implementación, de los cuales plasmaremos algunos en el desarrollo de este proyecto.

A. Programación Orientada a Objetos

Stroustrup [1] nos indica que la POO es una técnica o paradigma para escribir programas 'buenos' para un conjunto de problemas. Asímismo, menciona algunas características de las funciones de soporte principales:

- **Mecanismo de llamada:** Es el mecanismo mediante el cual se invoca una función miembro para un objeto, puede ser llamada por una funcion normal o virtual.
- **Comprobación de tipos:** Te permite invocar cualquier método para cualquier objeto.
- **Herencia:** Si un lenguaje no tiene algun mecanismo de herencia o tiene un mecanismo de herencia sin funciones o métodos virtuales, entonces este lenguaje no soporta la POO.
- **Herencia multiple:** Cuando una clase B hereda de una clase base A, la clase B siempre es una A además de cualquier otra cosa que pueda ser, entonces es posible que B herede de dos clases base, A1 y A2.
- **Encapsulamiento:** Para protegerse a una clase de acceso no autorizado, solo las operaciones definidas para esta clase o las funciones miembro pueden acceder a dicha clase.
- **Problemas de implementación:** La POO se basa en las mejoras de lenguaje ya proporcionadas para admitir la abstracción de datos, por lo que se necesitan relativamente pocas adiciones.

B. El juego del Ajedrez

Navarro [**?**] nos da una explicación bien detallada sobre el origen, las reglas básicas para colocar el tablero, las piezas, la forma en que estas se mueven y como se nombran a las jugadas del juego de Ajedrez:

1) ***Origen del Ajedrez:*** El Ajedrez se jugó por primera vez en China, India y Persa. La palabra 'Ajedrez' proviene del persa 'shah', que significa 'rey'. Fueron los europeos quienes dieron nombre a las piezas y los colores (blancas y negras) representando la forma en que se vivía en la Edad Media:

- Los peones (ocho) son los siervos, jornaleros y pobres que constituyen la infantería, y con frecuencia deben sacrificarse para proteger a las piezas más valiosas.
- La torre (dos) es la fortaleza, el refugio, el hogar.
- El caballo (dos) representa al único soldado profesional.
- Los alfiles (dos) son dignatarios episcopales de la iglesia, la parte superior de estas piezas se parecen a los sombreros usados por estos.
- La dama o reina es 'el poder detras del trono', sus movimientos combinan los del hogar (la torre) y la iglesia (el alfil).
- El rey es la pieza más importante (no la más poderosa) por representar la autoridad indiscutible, pero debe ser protegido porque su captura significa la pérdida de su reino.

*2) **Colocación del tablero y las piezas**:* Las casillas blancas de las esquinas del tablero deben quedar a la derecha de cada uno de los jugadores, el contendiente con las piezas blancas será quien haga la primera jugada. Las Piezas se ubican segun la Figura 1:

- Las piezas se disponen colocando a la reina en la casilla central de su color (reina blanca en casilla blanca, reina negra en casilla negra) en la columna o extremo del tablero frente al jugador.
- El rey se coloca en la casilla central adyacente a la de la reina. A ambos lados de los reyes se colocan, primero los alfiles, luego los caballos y en las esquinas del tablero las torres. Los peones se colocan como una barrera delante de las piezas mayores

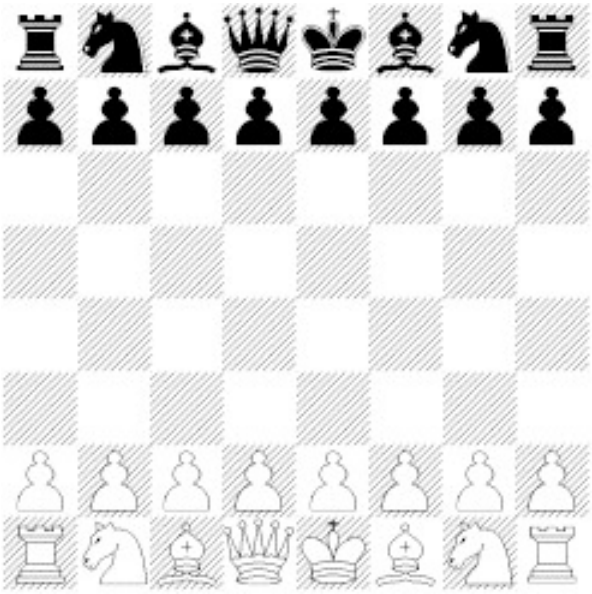

Fig. 1. Ubicación del tablero y las piezas

*3) **Movimientos y jugadas del Ajedrez**:*

- Los peones pueden avanzar una casilla hacia adelante cada vez, excepto en su primer movimiento en que opcionalmente pueden avanzar dos casillas.
- Las torres se mueven en linea recta en las direcciones horizontal o vertical (adelante, atras, izquierda o derecha).
- Los caballos se mueven dos casillas en cualquiera de las direcciones horizontal o vertical y una casilla en dirección perpendicular a la dirección del movimiento inicial (en forma de 'L'); es la única pieza del tablero que puede saltar piezas que se encuentren entre las posiciones inicial y final de su movimiento.
- Los alfiles se mueven en diagonal, hacia adelante o hacia atrás.
- La reina combina los movimientos de la torre y el alfil.
- El rey puede moverse solamente a cualquiera de las casillas adyacentes, o bien, enrocarse.
- **El enroque:** Se mueve al rey dos casillas hacia un lado, y luego se mueve la torre alrededor del rey. Para efectuar el enroque tanto el rey como la torre debe encontrarse en su posición de inicio de juego y no debe encontrarse pieza alguna entre ellas. Además, el enroque sólo puede efectuarse una vez y únicamente si el rey no está amenazado por alguna de las piezas contrarias ni tiene que moverse por una casilla amenazada por una pieza enemiga.
  Existen dos tipos de enroque: el corto (del lado del rey) y el largo (del lado de la dama)

II. Materiales

A. C++

- Un poco de Historia

Segun Stroustrup [**?**], creador del lenguaje C++, este lenguaje fue implementado y diseñado porque Stroustrup quería distribuir los servicios de un Kernel de UNIX a traves de multiprocesadores y redes de área local (lo que actualmente se conoce como multinúcleos y clústeres).

Tambien menciona el origen del nombre, C++, el cual inicialmente fué llamado C con Clases por su creador, mas tarde en el verano de 1983, Rick Mascitti acuña el nombre de C++, que significa la naturaleza evolutiva de los cambios de C ('++' es el operador de incremento de C).

- Justificacion de Uso

Usamos C++ 20 debido a que es un lenguaje para novatos o principiantes que quieran profundizar conceptos de POO, teniendo absoluto control de la memoria y los procesos.

B. Qt

- Justificacion de Uso

Usamos Qt 5.15.1 por que es un Framework que permite de manera sencilla diseñar y crear interfaces de aplicaciones de escritorio. Tambien lo integramos con Github para automatizar los cambios.

C. Git y Github

- Justificacion de Uso

Usamos Git mediante Gitbash para poder guardar y controlar los cambios del Proyecto de manera local y remota. Tambien usamos Github como repositorio en la nube, para mantener seguro, versionado y disponible nuestro Proyecto.

D. Materiales adicionales

1) CMake: Usamos CMake para configurar, integrar y automatizar Proyectos de Software, en nuestro caso lo integramos con Qt.

2) Overleaf: Usamos Overleaf por que nos permite trabajar de manera colaborativa y remota en la creación y edición de Artículos en distintos formatos, en nuestro caso optamos por el formato IEEE.

3) Google Meet: Usamos esta herramienta para poder realizar reuniones diarias, para poder coordinar mejor las responsabilidades y tareas a cumplir de cada integrante.

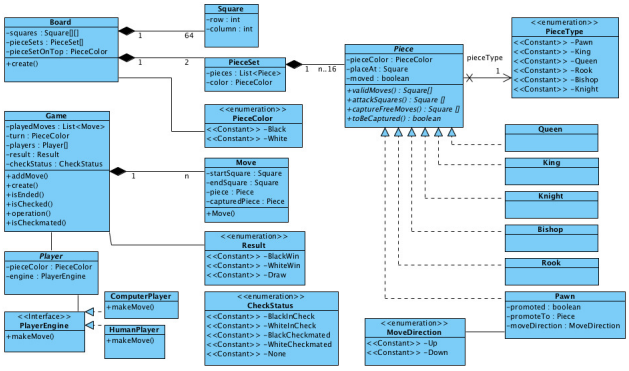

Fig. 2. Diagrama de clases

III. Métodos

A. Diagrama de Clases

La Figura 2 muestra nuestro diagrama de clases, con la cardinalidad y tipo de relación que establecimos inicialmente.

B. Explicacion de Técnicas de POO Implementadas

1) Singleton: Usamos Singleton en la clase **'Board'** para manejar una unica instancia de Board, que permita acceder a datos miembro y metodos de la instancia de esta clase de manera global. También usamos Sinleton en las clases InfoDetaisl y MainWindow. Por cada clase se genero una macro para cada instancia: **BOARD**, **INFODETAILS** y **MAINWINDOW**.

2) Polimorfismo: Usamos poliformismo en la clase **Piece** como clase base para el conjunto de piezas de ajedrez, es este caso las clases hijas son: **Pawn**, **Bishop**, **Rook**, **Knight**, **Queen** y **King**. Estas clases hijas se benefician de los datos miembro, metodos y funciones de la clase base que sobreescritos para posteriormente ser usados en la lógica del juego.

3) Herencia: Usamos herencia en la clase **Piece** y sus clases hijas, donde cada clase hija guarda sus datos e implementa su funciones de manera abstracta.

IV. El Juego: Guía de Usuario

- La pantalla inicial muestra datos de cada jugador (nombre de jugador, fichas comidas y un reloj).
- La primera jugada la realiza el jugador con las piezas blancas, pudiendo sus peones moverse hasta dos casillas en el primer movimiento que realicen.
- Todas las piezas a excepto del Peón, comen las fichas del jugador contrario mediante sus movimientos normales. El Peón es la única pieza que como con un movimiento distinto al que tiene para moverse, en este caso come las piezas en diagonal.
- Para un mejor entendimiento, revise la seccion de Movimientos y jugadas del Ajedrez.

V. Futuras actualizaciones

- **Enroque**, aún no se completo la implementación del enroque, por lo que será implementado en una futura versión.

```cpp
#ifndef SINGLETON_H
#define SINGLETON_H
#include <cassert>
#include <iostream>
template<typename Type> //Type es la clase
class Singleton
{
    static Type* Instance;
public:
    Singleton(){ }

    static Type& GetSingleton()
    {
        if(Instance == nullptr)
            Instance = new Type();
        return *Instance;
    }
    static Type* GetSingletonPointer()
    {
        if(Instance == nullptr)
            Instance = new Type();
        return Instance;
    }
    virtual ~Singleton()
    {
        Instance = nullptr;
    }

};

template <typename Type>
Type* Singleton<Type>::Instance = nullptr;

#define SINGLETON(T) Singleton< T >::GetSingleton()
#define SINGLETON_DESTROY(T) Singleton< T >::~Singleton()


#endif // SINGLETON_H
```

Fig. 3. Implementación del patron Singleton

```cpp
#ifndef PIECE_H
#define PIECE_H
#include "mylibrary_global.h"
#include "square.h"
class Piece
{
protected:
    std::string Color{};
    uint16_t ID{};
    int X{64};
    int Y{64};

    bool Captured{false};
    bool Moved{false};

    std::string UrlFigure;
public:
    Piece() = delete;
    Piece(std::string color, int x, int y): Color{color}, X{x}, Y{y} {};
    virtual std::string GetColor() const = 0;
    virtual uint16_t GetID() const = 0;

    virtual uint16_t GetX() const = 0;
    virtual uint16_t GetY() const = 0;
    virtual std::string GetUrlFigure() const = 0;
    virtual bool IsCaptured() const = 0;
    virtual bool IsMoved() const = 0;

    virtual std::vector<Square*> ValidMoves() const = 0;
    virtual std::vector<Square*> AttackSquares() const = 0;
    virtual std::vector<Square*> CaptureFreeMoves() const = 0;
    virtual void SetX(uint16_t x) = 0;
    virtual void SetY(uint16_t y) = 0;
    virtual void SetCaptured(bool captured) = 0;
    virtual void SetMoved(bool moved) = 0;
    virtual ~Piece(){/*std::cout<<"Piece destroyed\n";*/}
};

#endif // PIECE_H
```

Fig. 4. Clase base Pieza con implementación de polimorfismo

- **Material de diseño Badget**, se implementara una burbuja donde indique el número de fichas comidas no vistas al costado del nombre del jugador.
- **Reloj cronómetro**, este indicará el tiempo que demora el jugador en realizar su turno en el juego.
- **Señales de movimientos**, se podrá visualizar los movimientos posibles de cada ficha resaltando los colores de fondo de las casillas correspondientes.

VI. Conclusiones

- El patrón de diseño Singleton nos brinda una manera de gestionar la duplicidad de alguna de nuestras clases que requiera ser única, ademas de brindarnos un acceso global de la instancia Singleton.
- El juego de ajedrez es un excelente ejemplo para aplicar muchas caracteristicas del paradigma de POO, logrando así un aprendizaje más óptimo y de una manera más divertida.

References

[1] B. Stroustrup, "What is object-oriented programming?", IEEE software, 1988, vol. 5, no 3, pp. 10–20.

[2] M. Navarro, "El juego del ajedrez", Octubre 2000

[3] B. Stroustrup, The C++ Programming Languaje, 4th ed. Addison-Wesley, 2013.