Ciencia de la Computación

Ciencia de la Computación I

Docente Mamani Aliaga, Alvaro Henry

Informe del proyecto final de ciencia de la computacion I

Entregado el 27/11/2025

MAMANI ESCOBEDO, Jorge
CALLATA LAZO, Zorba
APAZA VISA, Piero Joel
TACORA CALLISAYA, Joham Joseph

Semestre II

2025-2

"Los alumnos declaran haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo"

_____ _____ _____ _____

CHRONO PATHFINDER

1.-Descripción del trabajo

ChronoPathfinder es un videojuego completo de género puzle-exploración creado desde cero utilizando exclusivamente C++ estándar y la biblioteca básica de entrada/salida (<iostream>, <fstream>). No se emplearon librerías gráficas externas (SFML, SDL, ncurses, etc.) ni contenedores modernos de la STL como std::vector, std::priority_queue o smart pointers.

El juego consta de varios niveles interconectados en los que el jugador debe llegar a una puerta de salida mientras gestiona un recurso escaso llamado Puntos de Historia (PH). Estos puntos representan literalmente "cuánta historia puedes reescribir".

La mecánica central y diferenciadora es la gestión de paradojas temporales:

- El jugador puede crear checkpoints (puntos de guardado temporales) gastando 1 PH.
- Al intentar avanzar a un nuevo nivel, el juego verifica automáticamente si en cualquiera de los niveles futuros existe un NPC marcado como "defensor" con el que no se tiene relación de amistad.
- Si se detecta dicha situación, se produce una paradoja temporal y el universo fuerza la carga del último checkpoint válido, perdiendo todo progreso desde ese punto.
- Para evitar la paradoja es necesario haber invertido previamente 2 PH en hacerse amigo del NPC correspondiente.

De esta forma surge un puzle de optimización y planificación temporal: el jugador debe decidir cuándo y en quién gastar sus preciados PH para no quedarse sin ellos antes de terminar el juego. El jugador que termina con más PH obtiene mejor posición en el ranking persistente.

El proyecto incluye implementación manual de estructuras de datos (vector dinámico y par ordenado), herencia, polimorfismo, sobrecarga de operadores, gestión estricta de memoria dinámica y un diseño altamente modular.

2.-Explicación de las funcionalidades

1. Movimiento y exploración

- Mapa fijo de 15 filas × 30 columnas.
- Teclas w/a/s/d para movimiento ortogonal.
- Colisión con bordes y detección automática de puerta de salida (>).

2. Sistema de NPCs y relaciones
   - Cada NPC tiene posición distinta por nivel y una relación natural (amistoso/neutral).
   - El jugador puede gastar 2 PH para convertir a un NPC en amigo permanente.
   - Las relaciones se guardan y restauran en los checkpoints.
3. Checkpoints y viaje en el tiempo
   - Función guardarCheckpoint(nivel, cantidadNPCs) almacena: PH actuales, nivel actual y vector completo de relaciones.
   - Función cargarCheckpoint() restaura todo el estado anterior.
   - Se llama automáticamente cuando se detecta una paradoja.
4. Detección de paradojas temporales
   - Método isCan() recorre todos los niveles desde el actual+1 hasta el final.
   - Para cada defensor de esos niveles comprueba si el jugador es amigo.
   - Si encuentra al menos un defensor no-amigo → devuelve false → no permite crear nuevo checkpoint ni avanzar.
5. Interfaz y menús
   - Menú principal con opciones: Iniciar juego – Puntajes – Salir.
   - Menú de pausa accesible con “p” (Continuar – Guardar checkpoint – Volver al menú – Salir del juego).
6. Persistencia de datos
   - Clase ScoreDB guarda los PH restantes al final del juego en “scores.txt”.
   - Muestra los 10 mejores puntajes (orden natural del archivo).
7. Estructuras de datos propias

    - Vec: vector dinámico con duplicación de capacidad, resize, reserve, operator[].
    - Pair: par ordenado con operadores <, >, == y << sobrecargados.
    - El operator< está invertido a propósito para usar std::priority_queue como min-heap en futuras implementaciones de A*.
8. Representación visual
    - Mapa ASCII actualizado en cada turno.
    - Jugador = “J”, NPCs = “N”, puerta = “>”, suelo = “.”.
9. Gestión de memoria
    - Todos los arreglos dinámicos (NPC**, Nivel**, bool*, Pair*, int* para relaciones) son liberados correctamente en los destructores.
    - No existen memory leaks (comprobado con Valgrind).

3.-Lecciones aprendidas

- Gestión manual de memoria en C++: new/delete, new[]/delete[], regla de los tres, destructores virtuales.
- Cómo evitar memory leaks en proyectos grandes.
- Uso de sobrecarga de operadores para que clases propias se comporten como tipos primitivos.
- Importancia de la separación de responsabilidades: lógica (Juego), presentación (Mapa/Menu), datos (Vec/Pair), entidades (Personajes/NPC/Jugador).
- Diseño de mecánicas de juego emergentes: con solo tres NPCs y dos niveles surgen docenas de estrategias distintas.
- Planificación temporal y optimización de recursos como elemento central de un puzle.
- Valor de la modularidad: añadir nuevos niveles o NPCs requiere únicamente modificar dos funciones de inicialización.
- Lectura y escritura segura de archivos en C++ .

4.-Conclusiones y recomendaciones

Conclusiones El proyecto ChronoPathfinder se entrega 100 % funcional, sin errores de ejecución ni fugas de memoria, con una mecánica de juego original y adictivo que combina conceptos de programación orientada a objetos con un diseño de juego creativo y profundo. A pesar de utilizar únicamente herramientas básicas de C++, se logró crear una experiencia completa con menús, guardado de puntuaciones, inteligencia de paradojas y una arquitectura limpia y extensible. La mecánica de gestión de PH y prevención de paradojas resultó ser extremadamente divertida y escalable, generando puzles no lineales interesantes incluso con pocos elementos.

ChronoPathfinder no es solo una tarea académica: es una prueba fehaciente de que con conocimientos sólidos de C++ y creatividad se pueden crear experiencias únicas y memorables.