

INTEGRADORA DE ALGORITMOS Y PROGRAMACIÓN II:

Presentado por:

- Alejandra Muñoz Ruiz
- Dayana Fernández Núñez
- Valentina Tangarife Rincón

Semana 10 - sábado 4 de octubre, Hora: 6:00 pm

- Especificación de requerimientos
- Diseños de pruebas
- Codificación de pruebas
- Diagramas de clase

Identificación del problema y análisis de requerimientos

Identificación del problema: El proyecto Oregon Trail Survival busca adaptar el clásico juego educativo *The Oregon Trail* a un entorno de acción y supervivencia en tiempo real, combinando gestión de recursos, exploración y combate. El problema identificado es la necesidad de diseñar un sistema que integre estas mecánicas dentro de un marco académico, utilizando estructuras de datos y conceptos de programación avanzada (listas enlazadas, árboles binarios, hilos, TDD), que permita al estudiante aplicar conocimientos técnicos en un caso práctico.

Cliente: La Facultad Barberi de Ingeniería, Diseño y Ciencias Aplicadas de la Universidad Icesi

Usuario: Jugador/es del videojuego

Requerimientos Funcionales:

Para el movimiento y exploración correctamente en el juego hemos identificado:

1. El sistema debe permitir al jugador mover al personaje en cuatro direcciones (arriba, abajo, izquierda, derecha) dentro de cada escenario.
2. El sistema debe restringir el movimiento del jugador con límites del mapa o paredes.
3. El sistema debe permitir la transición entre escenarios a través de puntos de acceso (puertas, pasajes montañosos, vados fluviales).

4. El sistema debe mostrar tres escenarios conectados que representen las etapas del viaje (Llanuras y praderas, Montañas Rocosas y Río Columbia).

Para el sistema de combate y armas:

5. El sistema debe permitir al jugador utilizar dos tipos de armas (rifle de avancarga y revólver), cada una con características de daño y velocidad.
6. El sistema debe mostrar una retícula de puntería controlada por el mouse para disparar.
7. El sistema debe gestionar la munición como un recurso limitado y restarla al disparar.
8. El sistema debe requerir recargar el arma después de vaciar el cargador.

Para la supervivencia y la gestión de recursos:

9. El sistema debe administrar la salud del jugador con un máximo de 3 impactos antes de morir.
10. El sistema debe permitir recolectar recursos del entorno (comida, medicinas y munición).
11. El sistema debe limitar el inventario del jugador para priorizar suministros esenciales.

Para la jugabilidad de autómatas enemigos:

12. El sistema debe generar autómatas enemigos de forma aleatoria en cada partida.
13. El sistema debe programar a los enemigos para perseguir y atacar al jugador.
14. El sistema debe aplicar daño al jugador cuando un enemigo lo ataque.
15. El sistema debe registrar la derrota del jugador cuando se reduzca su salud a cero.

16. El sistema debe permitir reiniciar o finalizar la partida tras la muerte del jugador.

Árbol de Logros y Estructuras de Datos:

17. El sistema debe implementar un árbol de logros utilizando un árbol binario de búsqueda, que muestre en una ventana independiente los logros alcanzados por el jugador.

18. El sistema debe almacenar en listas enlazadas todos los elementos susceptibles de gestión lineal (enemigos, recursos o inventario).

19. El sistema debe permitir visualizar un menú principal desde el cual se acceda a las diferentes pantallas del juego.

20. El sistema debe mostrar en pantalla indicadores del estado del jugador (vida, inventario y munición) durante la partida.

21. El sistema debe desplegar una pantalla de “Game Over” cuando el jugador pierda la partida.

22. El sistema debe desplegar una pantalla de victoria cuando el jugador complete el recorrido hasta Oregón.

23. El sistema debe mostrar el árbol de logros en una pantalla separada.

24. El sistema debe permitir consultar el manual de usuario dentro del juego a través de una ventana emergente (JavaFX).

Animaciones y Concurrencia

25. El sistema debe gestionar animaciones concurrentes para el movimiento de personajes y enemigos.

Integración con IA

26. El sistema debe integrar diálogos generados mediante la API de Gemini para enriquecer la interacción con enemigos y bots pasivos.

Seguimiento del Desarrollo (TDD y Métricas)

27. El sistema debe registrar indicadores de calidad del software (densidad de fallos, confiabilidad y completitud) en 15 commits equitemporales a lo largo del desarrollo.

Requerimientos no funcionales:

1. El proyecto debe entregarse en un repositorio de GitHub, con commits equitativos por cada integrante, evidenciando aportes individuales.
2. El desarrollo debe seguir la metodología TDD, implementando pruebas unitarias en JUnit para validar la lógica del sistema.
3. La interfaz gráfica debe desarrollarse con JavaFX en 2D, y las animaciones deben manejarse mediante hilos y concurrencia.
4. Todo elemento lineal susceptible de almacenamiento debe implementarse en listas enlazadas, y el árbol de logros debe ser un árbol binario de búsqueda.
5. El sistema debe mantener una estructura modular de paquetes en Java, separando la lógica del modelo (`model`), la interfaz de usuario (`ui`), el controlador (`controller`) y las pruebas (`test`).
6. El sistema debe integrar la API de Gemini para generar diálogos dinámicos con enemigos y bots pasivos.
7. La aplicación debe mostrar pantallas coherentes: menú principal, escenarios con indicadores de vida/inventario/munición, pantalla de victoria, pantalla de *game over* y ventana de árbol de logros.
8. Se debe incluir un manual de usuario accesible desde el juego, con controles, indicadores y resolución de incidentes.

9. La documentación final debe contener el diseño actualizado, trazabilidad de requerimientos, diagramas de clase y pruebas realizadas.

10. El proyecto debe entregarse de acuerdo con los plazos definidos:

- Semana 10 (4 de octubre, 6:00 pm): especificación de requerimientos, diseño de pruebas, codificación inicial de pruebas y diagramas de clase.
- Semana 14 (1 de noviembre, 6:00 pm): actualización de documentación y diseño, implementación de listas enlazadas y árboles, ordenamientos, búsqueda binaria y modelo en paquete `model`.
- Semana 18 (fecha final): entrega final y sustentación con interfaz 2D, animaciones, integración de modelo con la vista, diseño actualizado y documentación completa.

Formato de escenarios y casos de prueba

Configuración de los Escenarios

Nombre	Clase	Escenario
setupStage1		Juego iniciado en el Escenario 1 (Llanuras) con un jugador en posición inicial y sin enemigos
setupStage2		Juego iniciado en el Escenario 2 (Montañas Rocosas) con terreno difícil y un enemigo generado.
setupStage3		Juego iniciado en el Escenario 3 (Río Columbia) con recursos limitados, dos enemigos y jugador con 2 vidas restantes.

Diseño de Casos de Prueba

Objetivo de la Prueba: Verificar el movimiento del jugador y la transición entre escenarios.				
Clase	Método	Escenario	Valores de Entrada	Resultado esperado
Jugador	mover(Dirección)	setupStage1	Dirección = DERECHA	Jugador cambia su posición (x+1, y).
Jugador	estaVivo()	setupStage1	Ninguno	Retorna true (salud completa).
Inventario	agregarRecurso()	setupStage1	Recurso = comida	Comida añadida al inventario.
Inventario	mostrarInventario()	setupStage1	Ninguno	Muestra inventario con recurso comida.
SistemaJuego	iniciarPartida()	setupStage1	Escenario = Llanuras	Juego inicializado correctamente.

Objetivo de la Prueba: verificar sistema de combate y consumo de munición.

Clase	Método	Escenario	Valores de Entrada	Resultado esperado
Jugador	recibirDaño(int)	setupStage2	Daño = 1	Salud pasa de 3 → 2.
Jugador	atacar(Enemigo)	setupStage2	Rifle con munición, objetivo = enemigo	Enemigo recibe daño y puede ser eliminado.
Arma	recargar()	setuoStage2	Rifle vacío	Rifle vuelve a tener capacidad de disparar.
Inventario	agregarRecurso()	setupStage2	Recurso = medicina	Medicina añadida al inventario
Enemigo	atacarJugador()	setupStage2	Enemigo golpea al jugador	Jugador pierde 1 vida adicional.

Objetivo de la Prueba: Validar el comportamiento del juego en el Escenario 3 (Río Columbia) cuando el jugador tiene recursos limitados, dos enemigos activos y 2 vidas restantes, verificando la correcta gestión de salud, munición, recolección de recursos, ataques enemigos y finalización del juego (victoria o derrota).

Clase	Método	Escenario	Valores de Entrada	Resultado esperado
Jugador	mover(Dirección)	setupStage3	Dirección = ARRIBA	Jugador cambia de posición intentando cruzar río
Jugador	recibirDaño(int)	setupStage3	Daño = 1	Salud pasa de 2 → 1.
Jugador	estaVivo()	setupStage3	Salud = 0	Retorna false, jugador pierde la partida.
Arma	disparar(Enemigo)	setupStage3	Munición disponible = 1, objetivo enemigo	Enemigo recibe daño y puede ser eliminado.
Inventario	agregarRecurso()	setupStage3	Recurso = munición	Munición limitada añadida al inventario.
Enemigo	atacarJugador()	setupStage3	Enemigo golpea jugador con salud = 2	Jugador pasa a 1 vida restante.
SistemaJuego	actualizarEstado()	setupStage3	Jugador con 0 vidas o enemigos eliminados	Si jugador = 0 → fin del juego; si enemigos = 0 → victoria.