Proyecto Final

Paradigmas y Técnicas de Programación

3º iMAT

RobotEscape: The Lab Simulation

María Roman Cantillana

Irene Turrado Sierra

Contenido

Género:

Estrategia / Escape táctico / Simulación
Motor de desarrollo: Unity 6000.2.6f2
Plataforma objetivo: PC

Este juego pertenece al género de estrategia y simulación táctica**,** ya que el progreso del jugador depende de su capacidad para planificar y ejecutar movimientos precisos dentro del entorno del laboratorio. El jugador debe analizar la posición de los drones de seguridad, anticipar sus rutas y decidir en qué momento avanzar para alcanzar la salida sin ser detectado. Esto permite integrar sistemas de inteligencia artificial basados en estados, que dotan a los enemigos de comportamientos realistas y adaptados según la situación. Además, se sustenta una arquitectura modular que aplica patrones de diseño como *State*, *Singleton* y *Observer,* entre otros (se detalla más adelante).

Resumen general

RoboEscape: The Lab Simulation es un videojuego ambientado en un laboratorio futurista donde el jugador controla a un robot que intenta escapar del laboratorio antes de ser capturado por los distintos drones.
Durante la huida, el jugador deberá evitar ser detectado por distintos tipos de drones de vigilancia que patrullan las instalaciones. Cada dron tiene un comportamiento propio:

- **Scouts**: son rápidos y tienen una visión estrecha, detectan al jugador a distancia y realizan persecuciones breves.
- **Guardians**: son más lentos, pero con un amplio campo de visión, activan un estado de alerta global si detectan al jugador.
- **Sniffers**: están especializados en rastrear sonido, reaccionan ante ruidos generados por el movimiento o acciones del jugador.

Estos drones adaptan su comportamiento según el nivel de alerta y pueden coordinarse indirectamente a través de eventos de detección. Si cualquiera de ellos logra alcanzar al jugador, la partida finalizará.
El juego combina mecánicas de sigilo, detección visual, detección sonora e inteligencia artificial, creando una experiencia dinámica donde los enemigos reaccionan tanto al entorno como al comportamiento del jugador. Asimismo, el juego tiene un menú principal desde el cual el jugador puede seleccionar distintos escenarios (tipo de sala) y niveles de dificultad, que ajustan automáticamente la cantidad y tipo de drones presentes en la partida. El objetivo final es alcanzar la salida del laboratorio sin ser detectado ni atrapado.

Objetivo principal

El objetivo del jugador es escapar del laboratorio sin ser detectado ni alcanzado por los drones de seguridad que patrullan la zona.

Para ello, deberá desplazarse estratégicamente por el mapa, utilizando paredes, columnas y otros objetos del entorno que bloquean la visión de los drones para

mantenerse oculto. También deberá gestionar su velocidad para evitar generar ruido innecesario y analizar tanto el campo de visión como los patrones de movimiento de cada tipo de dron. La planificación del recorrido, el control del sonido y el uso adecuado de las sombras o zonas seguras serán esenciales para avanzar sin ser descubierto.

El jugador gana la partida al llegar a la puerta de salida sin ser detectado.
Pierde si un dron lo detecta visualmente o un Sniffer rastrea un ruido y lo alcanzan.

La dificultad seleccionada influye en la cantidad y tipo de drones, así como en la agresividad de sus patrones de búsqueda, aumentando el desafío del escape.

Mecánicas principales

1. Movimiento del jugador

- Control directo del robot mediante teclado. Tiene un movimiento fluido con capacidad de correr, andar, agacharse y gestionar la generación de ruido.
- Correr o interactuar con ciertos elementos produce sonido, lo cual puede alertar a los drones Sniffers.
- El jugador debe aprovechar coberturas (paredes, columnas, esquinas, etc.), sombras y ángulos muertos para evitar la detección.

2. IA de los enemigos (drones)

- Cada tipo de dron posee propiedades y comportamientos específicos que influyen en su capacidad de detección visual o sonora.
- La IA se adapta a la distancia, el ángulo de visión, los estímulos sonoros y los parámetros definidos por la dificultad de la partida, cumpliendo con los principios SOLID, especialmente Responsabilidad Única (SRP) y Abierto/Cerrado (OCP).
- Implementada mediante una máquina de estados finitos (FSM) modular. Los drones pueden cambiar entre los estados de Patrulla, Persecución, Búsqueda y, en algunas variantes, Alerta Global:
    - **Estado de Patrulla (Patrol State)**: es el estado inicial. El dron recorre una serie de waypoints predeterminados. El Scout patrulla rápido y con un campo visual estrecho. El Guardian patrulla lento, pero escanea grandes ángulos. El Sniffer patrulla atento a sonidos en lugar de depender del campo visual. Si se detecta visual o sonoramente al jugador, se realiza la transición al estado correspondiente.

- **Estado de Persecución (Chase State)**: el dron abandona su ruta de patrulla y se dirige al jugador. El Scout acelera y realiza persecuciones breves. El Guardian realiza persecuciones más lentas pero constantes, y puede activar Alerta Global, aumentando temporalmente la agresividad de todos los drones. El Sniffer sigue la ruta hacia la fuente del ruido o hacia la última posición confirmada. Si el dron pierde al jugador visual o sonoramente, entra en estado de Búsqueda.
- **Estado de Búsqueda (Search State)**: el dron explora el área de la última posición conocida del jugador. Realiza movimientos circulares o revisa puntos cercanos. El Sniffer amplía gradualmente su radio de rastreo. Si re-detecta al jugador, vuelve a Persecución. Si no, regresa a Patrulla tras un tiempo definido.
- **Estado adicional: Alerta Global (solo para Guardian)**: se activa cuando un Guardian detecta al jugador. Otros drones aumentan su velocidad, campo visual o sensibilidad al ruido. Se desactiva cuando pasan varios segundos sin detección.

3. Sistema de detección

- **Detección visual:** cada dron posee un campo de visión (menos los Sniffers, que solo escuchan) con ángulo y distancia configurables.
- **Detección sonora (solo Sniffers)**: correr, caer desde alturas o usar ciertos objetos genera ruido. Se crea un objeto temporal “NoiseTrigger”, que el Sniffer detecta mediante colisiones.

4. Gestión de partida

El juego sigue un **flujo cerrado** compuesto por las fases: **Menú principal → Partida → Victoria/Derrota → Reinicio o vuelta al Menú**.

Al comenzar, aparece un menú inicial donde el jugador puede configurar la partida seleccionando **el escenario** y **el nivel de dificultad**. Una vez confirmada esta configuración, se carga la escena correspondiente y comienza el juego.

Toda esta lógica está coordinada por un **GameManager**, que actúa como el controlador central del sistema. Su función es gestionar: la dificultad seleccionada, el escenario elegido, el inicio y fin de la partida, las condiciones de victoria o derrota, los cambios de estado global (como la alerta general), y la comunicación con la interfaz de usuario para mostrar mensajes y actualizar indicadores.

5. Puntuación

La puntuación final del jugador se determina combinando diversos factores que evalúan su rapidez y su capacidad de mantener el sigilo. El componente principal es el tiempo de escape, donde finalizar más rápidos otorga más puntos según el tiempo óptimo definido para cada escenario. A esto se suman las penalizaciones por detección, que varían en función del tipo de dron implicado: los Scouts aplican penalizaciones moderadas, los Guardians penalizan más, y los Sniffers añaden penalizaciones si rastrean ruido. Asimismo, completar escenarios con mayor dificultad aplica un multiplicador final que recompensa a los jugadores que eligen modos más exigentes. Cada escenario también incluye una pequeña bonificación temática (como evitar persecuciones o no generar ruido) que refuerza su estilo particular.

6. Interfaz de usuario (UI)

La interfaz de usuario (UI) presenta al jugador toda la información esencial durante la partida: el tiempo transcurrido, el estado de alerta del laboratorio, si existe detección activa o Alarma Global, así como el escenario y la dificultad seleccionados. Todos estos elementos se actualizan de forma dinámica a través de eventos gestionados por el GameManager, siguiendo un patrón Observer que garantiza que la UI refleje siempre el estado actual del juego.

Elementos técnicos destacados

- **IA modular y escalable (Patrón State)**

La inteligencia artificial de los drones se implementa mediante el patrón de diseño State, que organiza los comportamientos en clases independientes (Patrulla, Persecución, Búsqueda y Alerta Global). Este enfoque permite que: cada tipo de dron seleccione cómo interpreta cada estado, los Sniffers incorporen detección sonora sin afectar al resto, los Guardians activen estados globales sin romper la arquitectura, se puedan añadir nuevos estados sin modificar el código ya existente.

- **Gestor central único (Patrón Singleton)**

El GameManager utiliza el patrón Singleton para garantizar que exista una única instancia coordinando: el flujo principal de la partida, la carga de escenarios, la dificultad, los cambios de estado global y la interacción con la UI y el sistema de sonido.

- **Creación de enemigos dinámica y configurable (Patrón Factory)**

La generación de los drones se encapsula mediante un EnemyFactory, que permite instanciar diferentes tipos de drones, asignar sus parámetros de comportamiento,

ajustar la cantidad de enemigos según la dificultad seleccionada y cargar configuraciones específicas según el escenario

Gracias a la Factory, el sistema puede ampliarse fácilmente con nuevos tipos de drones sin modificar el código que los usa. Se potencia así el bajo acoplamiento y el cumplimiento del principio open close principle.

- **Sistema de comunicación desacoplado (Patrón Observer)**

La comunicación entre el GameManager, el UIManager, los sistemas de audio y los indicadores de alerta se implementa mediante el patrón Observer. Esto permite que la UI reaccione automáticamente a cambios, los elementos de sonido activen señales sin conocer la lógica interna del juego, los estados globales del laboratorio se propaguen sin dependencias directas.

- **Separación de responsabilidades del jugador (Patrón Component / Strategy)**

El jugador utiliza un conjunto de componentes independientes: movimiento, generación de ruido, interacción y detección de colisiones. Esta arquitectura se inspira en el patrón Component, utilizado de forma nativa en Unity, y permite modificar o ampliar las capacidades del jugador sin afectar a las demás.

- **Sistema de dificultad extensible (Strategy Light)**

El ajuste de dificultad sigue un enfoque similar al patrón Strategy, ya que cada dificultad define una estrategia de parámetros, el sistema intercambia configuraciones de forma transparente y no es necesario editar la lógica principal del juego. Esto facilita extender el juego con dificultades personalizadas o modos especiales.

Reinicio y progresión

Al finalizar una partida, el juego muestra una pantalla final que resume el resultado: “Escapaste del laboratorio” en caso de victoria, “Has sido detectado” en caso de derrota.

Esta pantalla incluye opciones para continuar la experiencia:

• **Play Again:** reinicia el juego con los parámetros anteriores.

• **Return to Main Menu:** devuelve al jugador al menú principal, donde puede: seleccionar un nuevo escenario, modificar la dificultad, iniciar una nueva partida desde cero.

• **Loop de juego cerrado:** Main Menu → Partida → Resultado (Win/Lose) → Reinicio o Menú → Nueva Partida

Historias de Usuario

HU1. Movimiento básico del jugador

Como jugador, quiero poder mover al robot en todas las direcciones usando el teclado para desplazarme libremente por el laboratorio.

HU2. Cámara de seguimiento

Como jugador, quiero que la cámara siga al robot desde una vista en tercera persona para mantener siempre una buena visibilidad del entorno.

HU3. Patrulla de los drones

Como jugador, quiero que los drones patrullen automáticamente entre varios puntos del mapa para simular vigilancia constante.

HU4. Detección visual del jugador

Como jugador, quiero que los drones detecten mi presencia si entro en su campo de visión, para que el juego resulte desafiante y realista.

HU5. Persecución del jugador

Como jugador, quiero que los drones me persigan activamente cuando me detecten, para sentir presión y dificultad durante la huida.

HU6. Pérdida de visión y búsqueda

Como jugador, quiero que los drones busquen mi última posición conocida si dejo de estar a la vista, para que su comportamiento sea coherente.

HU7. Condición de victoria

Como jugador, quiero ganar la partida si llego hasta la salida del laboratorio sin ser detectado, para cumplir el objetivo del juego.

HU8. Condición de derrota

Como jugador, quiero perder la partida si un dron me alcanza, para que exista una consecuencia directa a mis errores.

HU9. Sistema de puntuación

Como jugador, quiero que se calcule una puntuación basada en el tiempo de escape y en si he sido detectado o no, para medir mi rendimiento.

HU10. Interfaz de usuario (UI)

Como jugador, quiero ver en pantalla información sobre el tiempo, estado de alerta y mensajes de victoria o derrota, para entender en qué situación me encuentro.

HU11. Reinicio de partida

Como jugador, quiero poder reiniciar la partida desde el menú tras ganar o perder, para volver a intentarlo fácilmente.

HU12. Sonido y retroalimentación

Como jugador, quiero escuchar sonidos de alarma o alerta cuando un dron me detecte, para percibir de forma inmediata los eventos importantes.

HU13. Selección de escenario

Como jugador, quiero poder elegir entre distintos escenarios desde el menú principal, para variar la experiencia de juego y explorar mapas diferentes.

HU14. Selección de dificultad

Como jugador, quiero elegir el nivel de dificultad antes de empezar la partida, para ajustar la cantidad y comportamiento de los drones según mis preferencias.

HU15. Diferentes tipos de drones

Como jugador, quiero que existan distintos tipos de drones con comportamientos únicos (Scout, Guardian, Sniffer), para que el desafío sea más variado y estratégico.

HU16. Sistema de ruido del jugador

Como jugador, quiero que correr o realizar acciones genere ruido, para que mis decisiones de movimiento tengan consecuencias y deba gestionar mi sigilo.

HU17. Detección sonora (Sniffer)

Como jugador, quiero que ciertos drones puedan detectar los sonidos que produzco, para que deba evitar correr o activar mecanismos ruidosos cuando estén cerca.

HU18. Alerta global

Como jugador, quiero que algunos drones activen un estado de alerta general cuando me detecten, para aumentar la tensión y dificultad temporalmente.

HU19. Feedback visual del estado del dron

Como jugador, quiero que los drones muestren cambios visuales (color o luces) según su estado (patrulla, persecución, alerta), para entender rápidamente su comportamiento.

HU20. Feedback del ruido

Como jugador, quiero ver un indicador visual cuando estoy generando ruido, para saber cuándo es más seguro caminar o correr.

HU21. Retorno al menú principal

Como jugador, quiero poder volver al menú principal desde la pantalla de derrota o victoria, para cambiar de escenario o dificultad sin cerrar el juego.

Análisis de Historias de Usuario

HU1. Movimiento básico del jugador: Se busca permitir al jugador moverse libremente por el laboratorio. Se crea el script PlayerController, se configuran teclas (WASD o flechas), velocidad, aceleración y colisiones, y se vincula la cámara. El movimiento debe ser fluido, continuo y con colisiones precisas, sin atravesar paredes y manteniendo 60 FPS. El jugador se desplaza y evita obstáculos con el teclado. Usa PlayerController y CameraFollow. Se valida que no haya retardo ni fallos. Prioridad: Alta.

HU2. Cámara de seguimiento: Objetivo: mantener siempre visible al jugador con cámara en tercera persona. Se desarrolla CameraFollow, ajustando distancia, suavizado y evitando giros bruscos. Debe centrarse en el jugador, sin vibraciones ni atravesar paredes. Cuando el jugador gira, la cámara se adapta. Diseñada con CameraFollow dependiente de PlayerController. Pruebas: seguimiento estable. Prioridad: Media-alta.

HU3. Patrulla de los drones: Los drones patrullan rutas fijas simulando vigilancia. Se implementa PatrolState, se definen waypoints y se controla velocidad y rotación. Deben patrullar automáticamente sin chocar. Un dron recorre A-B-C y vuelve. Usa EnemyAIController con patrón State. Se valida que no se atasque. Prioridad: Alta.

HU4. Detección visual del jugador: Los drones deben detectar al jugador en su campo de visión. Se crea CanSeePlayer(), con ángulo, rango y raycast para e vitar detección a través de muros. Precisión y reacción inmediata. Sin falsos positivos. Caso: jugador entra al cono → alarma. Usa EnemyAIController (State Pattern). Pruebas: detección correcta. Prioridad: Alta.

HU5. Persecución del jugador: Cuando el jugador es visto, el dron lo persigue. Se desarrolla ChaseState, con movimiento hacia el jugador, distancia y temporizador. Debe moverse fluido, sin atravesar objetos. Caso: dron sigue al jugador hasta que se oculta.

Usa EnemyAIController + ChaseState. Validar transición Patrulla→Persecución. Prioridad: Alta.

HU6. Pérdida de visión y búsqueda: Si el dron pierde de vista al jugador, busca su última posición. Se implementa SearchState con memoria y temporizador ≤ 5 s. Se mueve a la última ubicación y vuelve a patrullar tras el tiempo. Usa EnemyAIController. Se prueba transición Persecución→Búsqueda→Patrulla. Prioridad: Media-alta.

HU7. Condición de victoria: El juego acaba si el jugador alcanza la salida sin ser visto. Se crea zona ExitZone (BoxCollider) con OnTriggerEnter() y GameManager.WinGame(). Requiere detección precisa y sólo válida si no fue detectado. Caso: llega sin ser visto → "Victoria". Usa GameManager + UIManager. Se valida activación del panel. Prioridad: Alta.

HU8. Condición de derrota: La partida termina si un dron atrapa al jugador. Se añaden Colliders y OnCollisionEnter() que llama a GameManager.LoseGame(). Debe finalizar sin errores ni colisiones falsas. Caso: dron alcanza jugador → derrota. Usa GameManager + UIManager. Se comprueba bloqueo del movimiento. Prioridad: Alta.

HU9. Sistema de puntuación: Registra el rendimiento del jugador. Se crea ScoreSystem que calcula tiempo y penalizaciones y muestra la puntuación final. Debe ser preciso y coherente. Caso: jugador gana → puntuación según tiempo y detección. Usa ScoreSystem + UIManager. Se valida puntuación exacta. Prioridad: Media.

HU10. Interfaz de usuario (UI): Muestra información durante la partida. Se diseña HUD con cronómetro, alertas y mensajes, conectado al GameManager. Debe actualizarse en tiempo real. Caso: alerta roja al ser detectado. Usa UIManager (Observer). Se prueba cambio automático según estado. Prioridad: Media-alta.

HU11. Reinicio de partida: permite repetir la partida tras ganar o perder. Se añade botón "Play Again" y RestartGame() en GameManager. Debe reiniciar limpiamente la escena. Caso: jugador pulsa "Reintentar". Usa GameManager (Singleton). Se prueba reinicio sin errores. Prioridad: Media.

HU12. Sonido y retroalimentación: aumenta la inmersión con sonidos asociados a eventos. Se añaden AudioSource y se sincronizan efectos (alarma, victoria, derrota). Caso: alarma al detectar jugador. Usa SoundManager + GameManager (Observer). Se valida sonido correcto en cada evento. Prioridad: Baja-media.

HU13. Selección de escenario: esta historia añade variedad y rejugabilidad al permitir elegir diferentes mapas desde el menú principal. El sistema debe cargar escenas distintas sin modificar la lógica base de IA ni del jugador. A nivel técnico implica gestionar una lista de escenarios y un botón que invoque al SceneManager. Impacta en la estructura del menú y la organización de escenas en Unity.

HU14. Selección de dificultad: la ificultad modifica la cantidad y el comportamiento de los drones, por lo que requiere un sistema parametrizable antes de comenzar la partida. Implica integrar el DifficultyManager con el GameManager y ajustar velocidades, FOV y número de enemigos. Aumenta la flexibilidad del juego sin cambiar la lógica interna.

HU15. Diferentes tipos de drones: introduce variedad en la IA mediante comportamientos especializados (Scout, Guardian, Sniffer). Requiere herencia o composición sobre una clase base EnemyAIController y adaptación del patrón State según el tipo. Aumenta el desafío y obliga al jugador a emplear estrategias distintas según el dron.

HU16. Sistema de ruido del jugador: El ruido añade una mecánica nueva de sigilo. Implica que el jugador genere "NoiseTriggers" al correr o ejecutar acciones ruidosas. La implementación debe ser ligera y coherente con el motor físico de Unity. Incrementa la profundidad del juego y condiciona la forma de moverse.

HU17. Detección sonora (Sniffer): permite que los drones Sniffer reaccionen a estímulos sonoros. Requiere leer los NoiseTriggers generados por el jugador e integrarlos dentro de la FSM como transición al estado de Búsqueda. Aumenta el realismo del sistema de IA y obliga al jugador a controlar su ruido.

HU18. Alerta global: Cuando un Guardian detecta al jugador, se activa un estado global que afecta a todos los drones (aumento de velocidad, visión o agresividad). Exige una comunicación centralizada mediante GameManager y uso del patrón Observer para notificar el cambio. Aumenta la tensión y crea picos de dificultad.

HU19. Feedback visual del estado del dron: el jugador necesita reconocer rápidamente el estado de los drones (patrulla, persecución, alerta). Requiere cambiar el color o luces del dron según el estado de su FSM. Mejora la claridad visual y reduce frustración, sin modificar el comportamiento, solo su representación.

HU20. Feedback del ruido del jugador: indica visualmente cuándo el jugador está generando ruido. Informa al usuario sobre el riesgo de ser detectado por drones Sniffer. Aumenta la accesibilidad y facilita decisiones tácticas.

HU21. Retorno al menú principal: permite volver al menú desde la pantalla de victoria/derrota para cambiar escenario o dificultad. Mejora el loop de juego y evita reiniciar la aplicación o cargar manualmente.

Reflexión sobre el uso de herramientas de IA

Durante el desarrollo de este proyecto se utilizaron herramientas de inteligencia artificial, de manera complementaria y responsable. Su uso estuvo orientado a aclarar conceptos técnicos, resolver dudas concretas relacionadas con patrones de diseño, arquitectura de software y organización del código, así como a detectar posibles incoherencias en la documentación asociada al diseño del sistema.

La IA también se empleó de forma puntual para depurar errores, mejorar la redacción y asegurar la coherencia estructural del documento, siempre contrastando la información con el contenido de clase, la documentación oficial y el análisis propio realizado durante el proyecto. En ningún caso sustituyó el trabajo de diseño, razonamiento o toma de decisiones técnicas, sino que actuó como herramienta de apoyo al aprendizaje, permitiendo una comprensión más profunda de ciertos elementos del diseño y facilitando la elaboración de un informe claro y bien estructurado.