UASLP
Universidad Autónoma de San Luis Potosí

FACULTAD DE INGENIERÍA

Área de Ciencias de la Computación

Manual de Programador – Juego SkyRunner

Ana Laura Martínez Nava

Jimena Anais Sánchez Espino

Diego Emilio Ibarra Villela

Materia: Tecnología Orientada a Objetos

Profesor: Dr. Francisco Javier Torres

Fecha de entrega: 03 de diciembre de 2025

Semestre: 2025-2026/I

Introducción.

SkyRunner es un videojuego 2D tipo side-scroller desarrollado en Java, donde el jugador controla a un personaje aéreo que debe esquivar obstáculos, derrotar enemigos y avanzar a través de distintos mundos flotantes.

El entorno del juego es dinámico: se generan nubes, rayos, turbinas, enemigos y power-ups mientras el jugador avanza.

La arquitectura del proyecto fue diseñada totalmente con programación orientada a objetos, lo que permite mantener un control claro sobre cada elemento del juego y su interacción.

El motor se basa en un *Timer* que actualiza el estado del juego y redibuja la pantalla varias veces por segundo. Gracias a esto, se obtiene un movimiento fluido tanto del personaje como de los objetos del mundo.

Diseño de Clases y su Interacción.

El diseño general del sistema sigue una arquitectura orientada a objetos que se caracteriza por una jerarquía definida. En la base de esta estructura se encuentra la clase *GameObject*, que establece una interfaz común con los métodos y propiedades esenciales para cualquier entidad del juego. A partir de esta clase base, todas las demás entidades, como el personaje del jugador, los enemigos o los obstáculos, heredan y especializan su comportamiento, sobreescribiendo los métodos necesarios para definir su lógica específica y actualización.

La coordinación central de la aplicación recae en la clase *SkyRunnerGame*, que actúa como el núcleo principal orquestando el ciclo de juego. Las principales clases y su función dentro del sistema son:

1. SkyRunnerGame (clase central del juego)

- Controla todo el flujo del juego: estados, menú, pausa, game over.
- Administra la creación de obstáculos, enemigos, nubes, partículas y power-ups.
- Recibe la entrada del usuario (teclas).
- Gestiona colisiones y actualizaciones por frame.
- Encargada del renderizado total del juego (fondos, jugador, UI).
- Métodos principales:
    - SkyRunnerGame(): configura panel, fondo, foco y llama a initGame
    - initGame(): crea Aero, LevelManager, SoundManager, listas de entidades, ajusta velocidades base, reinicia puntuación y genera nubes iniciales
    - generateInitialClouds(): pre-puebla el cielo con nubes decorativas
    - generateInitialWorldFeatures(): agrega elementos temáticos al fondo según mundo
    - actionPerformed(ActionEvent): tick del timer; si se está jugando actualiza y repinta
    - updateGame(): ciclo completo de actualización del juego
    - updatePlayer(): aplica entrada, movimiento, turbo y estado
    - generateGameObjects(): genera nubes, obstáculos, enemigos, power-ups y rasgos
    - updateObjects(): desplaza y anima todos los objetos del juego
    - checkCollisions(): detecta colisiones con obstáculos, enemigos, proyectiles y power-ups
    - cleanupObjects (): elimina objetos inactivos o fuera de pantalla

- updateScore (): suma puntos por distancia y objetos esquivados
- checkGameConditions(): GAME_OVER si no hay vidas
- updateSounds (): manejo dinámico del motor y turbo
- paintComponent (Graphics): dibuja fondo, juego, UI y pantallas de fin
- drawBackground (Graphics): renderiza fondo y nubes
- drawGame (Graphics): dibuja entidades del juego
- drawUI (Graphics): barra de turbo, vidas, progreso y puntuación
- drawGameOver (Graphics): pantalla final
- drawVictory (Graphics): pantalla de victoria
- keyPressed(KeyEvent),handleGameKeyPress(int),handleEndGameKeyPress(int), keyReleased(KeyEvent)
- setDifficulty(String),setWorld(String),setSoundEnabled(boolean), setVolume(float), startGame(), pauseGame(), restartGame(), returnToMenu()
- Getters: estado, puntuación y distancia

2. Aero (jugador)

o Personaje controlado por el usuario.

o Atributos principales: posición, velocidad, salud, turbo, invulnerabilidad.

o Puede disparar, moverse en 4 direcciones y recibir power-ups.

o Se comunica directamente con SkyRunnerGame.

o Métodos principales:

- Aero(int,int): inicializa nave
- update (): movimiento, animación y estados
- draw (Graphics2D): dibuja nave y propulsión
- drawHealthBar (Graphics2D)
- moveUp/Down/Left/Right ()
- shoot (): dispara si no está en cooldown
- takeDamage (), takeDamage(int), applyDamage(int)
- heal(int)
- activateTurbo (), activateShield()
- reset ()
- getBounds (): caja de collision
- setInvulnerable(int), setLives(int)
- applySlow(double,int)
- isTurboActive (), isInvulnerable (), isShieldActive ()

3. Enemy

o Representa enemigos de diferentes tipos.

o Algunos disparan proyectiles, otros se mueven a velocidades distintas.

o Poseen salud, daño y comportamiento específico.

o Métodos principales:

- Constructores
- update (): animación y oscilación

- update(int): desplazamiento con scroll
- draw (Graphics2D)
- canShoot (), shoot ()
- getBounds ()
- takeDamage (int)
- getHealth ()

4. Obstacle

o Objetos que deben evitarse:

- Turbinas que empujan.
- Rayos (columnas eléctricas).

o Objetos sólidos.

o Métodos principales:

- Constructores
- update (), update (int)
- draw (Graphics2D)
- drawRockTower, drawElectricStorm, drawTurbine, drawCityBuilding
- isHarmful ()
- getType()
- getLightningColumnBounds(int)
- applyEffect(Aero)
- getBounds()

5. PowerUp

- Objetos que benefician al jugador:

  - Curación (vida extra)
  - Turbo
  - Escudo temporal
  - Velocidad

- Métodos principales:

  - PowerUp(int,int,Type)
  - update(), update(int)
  - getBounds()
  - applyEffect(Aero)
  - draw(Graphics2D)
  - drawSymbol(Graphics2D)
  - drawStar(...)
  - drawParticles(Graphics2D)
  - collect(Aero)

6. Projectile

- Representa los proyectiles del jugador y enemigos.

- Su movimiento es lineal.

- Desaparecen cuando salen de la pantalla o impactan algo.

- Métodos principales:

    - Projectile(int,int,int,boolean)
    - update()
    - draw(Graphics2D)
    - isEnemyProjectile()
    - getBounds()
    - getSpeed()

7. LevelManager

- Controla el progreso del nivel.

- Lleva el conteo de vidas.

- Determina si el nivel se completó o si el jugador perdió.

- Métodos principales:

    - LevelManager(String,String)
    - generateLevels(String,String)
    - getLevelName(String,int)
    - advanceLevel()
    - isGameComplete()
    - generateObstacle(int,int)
    - generateEnemy(int,int)

- generatePowerUp(int,int)
- loseLife(), addLife(), isGameOver()
- update(int)
- enemyDefeated(), obstaclePassed()
- getProgressInfo()
- getFinalScore(), getFinalStats()

8. SoundManager

o Reproduce música y efectos de sonido.

o Contiene funciones para:

- Disparo
- Explosiones
- Daño
- Power-ups
- Música ambiental por mundo

o Métodos principales:

- playShootSound
- playExplosionSound
- playPowerUpSound
- updateEngineSound
- playBackgroundMusic
- stopBackgroundMusic

- playWorldMusic

9. WorldFeature / Cloud

o Elementos del fondo con efecto parallax.

o Dan sensación de profundidad y movimiento.

o Métodos principales:

- Update
- Draw
- isOffScreen

o Flujo del Juego:

- Inicialización en el menú principal
- Inicio del juego al presionar Start
- Ciclo continuo con updateGame() y repaint()
- Generación de enemigos, obstáculos y power-ups
- Detección de colisiones
- Gestión del progreso del nivel
- Finalización por victoria o Game Over

Tiempo invertido por integrante.

- Diego Emilio

  - Funciones realizadas: Motor del juego, mecánicas del jugador, colisiones, obstáculos avanzados, integración total del proyecto
  - Horas Invertidas: 24

- Ana Laura

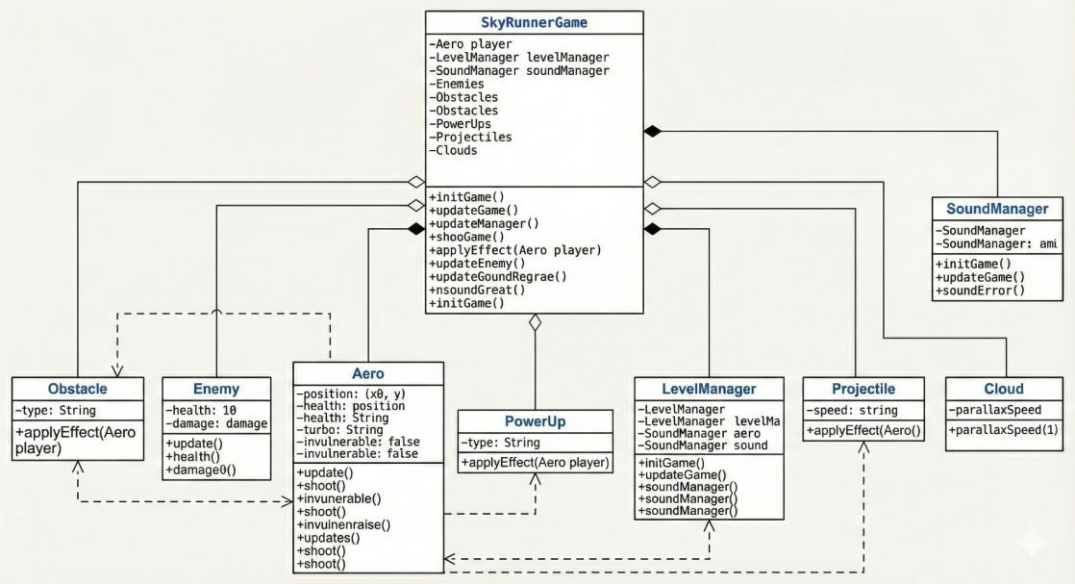
  - Funciones realizadas: Primeras versiones del juego, Diseño visual, sprites, animaciones, fondos y pantalla, power-ups, obstáculos avanzados, balance del nivel
  - Horas Invertidas: 20

- Jimena Anais

  - Funciones realizadas: Música del mundo, efectos de sonido y sistema, SoundManager, diseño visual
  - Horas Invertidas: 1

Diagrama UML

El diagrama de clases UML (Lenguaje Unificado de Modelado) modela la estructura estática del sistema *SkyRunner*, detallando las clases, sus atributos, sus métodos y las relaciones entre ellas.



Problemas y cómo los resolvimos:

- Problema 1: El juego se congelaba al iniciar

- Causa: el Timer no comenzaba después de generar los objetos.
- Solución: mover gameTimer.start() a startGame().

- Problema 2: El jugador atravesaba algunos obstáculos

- Causa: las hitboxes no coincidían con los sprites.
- Solución: corregir getBounds() en jugador, obstáculos y enemigos.

Conclusiones.

SkyRunner es la prueba de que, con una buena organización y un diseño claro, se pueden crear experiencias digitales complejas y entretenidas sin que el código se convierta en un caos. El juego demuestra cómo separar sus diferentes partes (el personaje, los enemigos, la música, los niveles) hace que todo sea más fácil de entender, modificar y mejorar. Esta forma de trabajar no solo ayudó a que el desarrollo fuera más ágil, sino que también asegura que, si en el futuro se quieren añadir nuevos elementos o mundos, el proceso será directo y sin complicaciones.

La experiencia de juego fluida que se consiguió con movimientos suaves, colisiones precisas y un audio que reacciona a lo que sucede en pantalla, es el resultado directo de haber pensado cuidadosamente en la estructura desde el principio. Cada elemento tiene su lugar y su función, trabajando en conjunto de manera armónica y esto hizo posible manejar múltiples acciones simultáneas sin que el rendimiento se resintiera.

En conclusión, este proyecto no solo cumple con su objetivo de entretener, sino que también deja un modelo valioso para enfrentar desafíos de desarrollo futuros, mostrando que lo bien construido perdura y se adapta con el tiempo.