**1. 📅 Hoja de Ruta por Fases con Entregables**

| **Fase** | **Duración Estimada** | **Objetivos Principales** | **Buenas Prácticas Clave** | **Entregable** |
| --- | --- | --- | --- | --- |
| **1. Núcleo Jugable** | 2–3 semanas | - Configurar proyecto Gradle (core + desktop).- Implementar ECS mínimo: Entity, Component y System.- Grid 2D + renderización de tokens.- Consola de comandos básica (/roll, /spawn, /describe). | - Separar paquetes: core.logic, core.render, core.ui.- AssetManager asíncrono.- Pooling de entities.- Programar contra interfaces. | Demo básica con mapa, tokens y consola. |
| **2. Modo Edición** | 2 semanas | - Implementar Command Pattern (undo/redo).- Colocación dinámica de tiles y tokens en pantalla.- EventBus desacoplado para edición. | - Sistemas aislados (EditingSystem vs GameSystem).- Publicar eventos de cambio de mundo únicamente desde UI.- Tests unitarios. | Editor in-game funcional con deshacer/rehacer. |
| **3. Persistencia y JSON** | 1 semana | - Serializar/deserializar mapas y entidades usando Jackson o Moshi y record Java.- Crear fixtures de prueba. | - Componentes inmutables (record).- IDs con UUID.- JSON legible y versionable.- Cobertura de tests para I/O. | Módulo io con carga/guardado funcionando. |
| **4. UI y HUD** | 1–2 semanas | - Diseñar pantallas y menús con Scene2D.UI.- Skin + TableLayout para consola y botones.- Manager de pantallas (StageManager). | - Listeners desacoplados que emitan CommandIssuedEvent.- UI reactiva: separar lógica de negocio.- Documentar skins y estilos. | Interfaz mínima con botones y consola interactiva. |
| **5. Eventos e IA** | 2 semanas | - Arquitectura de eventos global (EventManager).- Integración inicial con mock de IA (IAService). | - Inyección de dependencias para IA.- Circuit breaker y cache de prompts.- Logs de trazas para debugging de IA. | Plugin IA simulado que responde comandos. |
| **6. Multiplayer básico** | 3 semanas | - Cliente/servidor con libGDX-net o Netty.- Autoridad del servidor.- Sincronización de snapshots. | - Validar comandos en servidor.- Interpolación de estado en cliente.- Autenticación básica (token). | Demo LAN con DM + varios jugadores sincronizados. |

**2. 🏷️ Mejores Prácticas Revisadas**

1. **Modularidad y Gradle Multi-módulo**
   * Separar al menos tres módulos:
     + core: lógica de juego, ECS, eventos, IO.
     + desktop: launcher Java SE.
     + server (opcional): lógica de servidor y validación.
   * Cada módulo con su propio build.gradle, dependencias estrictas.
2. **Programación contra Interfaces**
   * Definir interfaces para servicios críticos:
   * public interface IAService {
   * String generarDescripcion(String contexto);
   * }
   * Inyectar implementaciones (mock en tests, real en producción).
3. **Inmutabilidad y Datos Puros**
   * Usar record para todos los componentes de ECS:
   * public record Position(int x, int y) {}
   * Facilita serialización y evita bugs por mutación inesperada.
4. **AssetManager y Carga Asíncrona**
   * Siempre cargar texturas y skins de manera asíncrona:
   * assetManager.load("ui/skin.atlas", TextureAtlas.class);
   * assetManager.finishLoading(); // Sólo al comienzo
5. **EventBus Desacoplado**
   * Centralizar el despacho de eventos. Ningún sistema debe referenciar directamente otro.
   * Ejemplo de registro:
   * eventBus.register(RollCommandEvent.class, event -> logicSystem.handleRoll(event));
6. **Testing y CI/CD**
   * Tests unitarios en cada módulo (core y io).
   * Workflow en CI: compile → tests → empaquetado → ejecutable desktop.
   * Incluir cobertura mínima del 80%.
7. **Documentación y Versionado**
   * JavaDoc en puntos críticos (ECS, EventBus, IO).
   * README.md con diagramas (Mermaid) y pasos de arranque.
   * Convención de nombres para assets: tile\_grass\_01.png, entity\_goblin\_01.png.
8. **Profiling y Optimización**
   * Integrar gdx-inspector para medir draw calls, uso de texturas.
   * Pooling de objetos en sistemas de lógica y render.

**3. 📂 Esquema de Carpetas y Módulos**

project-root/

├── android/ # Módulo Android (opcional)

├── core/

│ ├── src/main/java/com/mygame/core/

│ │ ├── ecs/

│ │ │ ├── Entity.java

│ │ │ ├── Component.java

│ │ │ └── system/

│ │ │ ├── RenderSystem.java

│ │ │ └── LogicSystem.java

│ │ ├── components/

│ │ │ ├── PositionComponent.java

│ │ │ └── StatsComponent.java

│ │ ├── event/

│ │ │ ├── EventBus.java

│ │ │ ├── GameEvent.java

│ │ │ └── RollCommandEvent.java

│ │ ├── io/

│ │ │ ├── PersistenceService.java

│ │ │ └── JsonMapper.java

│ │ └── service/

│ │ ├── IAService.java

│ │ └── IAServiceImpl.java

│ └── src/main/resources/

│ ├── assets/

│ └── ui/

├── desktop/

│ └── src/main/java/com/mygame/desktop/

│ └── DesktopLauncher.java

├── server/ # Lógica y validación servidor

│ └── src/main/java/com/mygame/server/

└── build.gradle

* **core/**: lógica común y reutilizable en desktop, android, html.
* **desktop/**: punto de entrada de la aplicación en JVM.
* **server/**: (opcional) motor de juego autoritativo y validaciones.

**4. ⚙️ Arquitectura ECS y Sistema de Eventos**

**4.1. Diagrama de Clases ECS**

classDiagram

class Entity {

<<record>>

UUID id

String name

Map<Class<? extends Component>, Component> components

}

class Component <<interface>>

class PositionComponent {

+int x

+int y

}

class RenderComponent {

+String spritePath

}

class System <<interface>>

class RenderSystem {

+void update(float delta)

}

class LogicSystem {

+void update(float delta)

}

Entity "1" o-- "\*" Component

RenderSystem ..> RenderComponent

LogicSystem ..> PositionComponent

* **Entity**: contenedor de componentes.
* **Component**: datos puros (record).
* **System**: lógica que opera sobre entities con ciertos componentes.

**4.2. Diagrama de Flujo del Bucle Principal**

flowchart TD

A[Inicio] --> B[Cargar Mundo (JSON)]

B --> C[Bucle de Juego]

subgraph Frame

C1[1. Procesar Input & UI] --> C2[2. Despachar Eventos]

C2 --> C3[3. LogicSystem.update()]

C3 --> C4[4. EditingSystem.update()]

C4 --> C5[5. RenderSystem.update()]

C5 --> C6[6. uiStage.act() y draw()]

C6 --> C7{¿Autosave?}

C7 -- Sí --> C8[Guardar Mundo]

C8 --> C1

C7 -- No --> C1

end

C --> D{¿Juego activo?}

D -- Sí --> C

D -- No --> E[Fin y Serializar]

* **Autosave** cada N frames o tras comando de edición.
* **EventBus** conecta UI y systems de lógica.

**5. 🎨 Uso de Scene2D.UI en libGDX**

1. **Inicialización de Stage y Skin**
2. Stage uiStage = new Stage(new ScreenViewport());
3. Skin skin = new Skin(Gdx.files.internal("ui/skin.json"));
4. Gdx.input.setInputProcessor(uiStage);
5. **Creación de Widgets y Layout con Table**
6. TextButton rollBtn = new TextButton("Roll d20", skin);
7. rollBtn.addListener(new ChangeListener() {
8. @Override
9. public void changed(ChangeEvent event, Actor actor) {
10. eventBus.fire(new RollCommandEvent("d20"));
11. }
12. });
13. Table table = new Table();
14. table.top().left().pad(10);
15. table.add(rollBtn).width(120).height(40);
16. uiStage.addActor(table);
17. **Render Loop Integrado**
18. @Override
19. public void render(float delta) {
20. // 1) Update lógica del juego
21. gameWorld.update(delta);
22. // 2) Dibujar mundo
23. gameWorld.render();
24. // 3) UI
25. uiStage.act(delta);
26. uiStage.draw();
27. }
28. **Cambio de Pantallas**
29. public class StageManager {
30. private final Game game;
31. public void setScreen(Screen screen) {
32. game.setScreen(screen);
33. }
34. }

* Cada pantalla (Screen) contiene su propio Stage y layout.
* Los listeners de UI sólo disparan eventos: **nunca** ejecutan lógica directamente.

**6. 🔄 Diagramas en Texto**

**6.1. Diagrama de Flujo de Comando /roll**

flowchart LR

UI[Botón Roll d20] -->|click| Evt[RollCommandEvent("d20")]

Evt --> Bus[EventBus.fire()]

Bus --> Logic[LogicSystem.handleRoll()]

Logic --> Dice[RollService.lanzar("d20")]

Dice --> Logic[LogicSystem.publicar(RollResultEvent)]

Logic --> UIConsole[ConsoleSystem.mostrar("Resultado: X")]

**6.2. Diagrama de Clases Simplificado**

classDiagram

class RollCommandEvent {

+String notation

}

class RollResultEvent {

+int value

}

class RollService {

+int lanzar(String notation)

}

class LogicSystem {

+void handleRoll(RollCommandEvent)

+void onRollResult(RollResultEvent)

}

RollCommandEvent --> EventBus

EventBus --> LogicSystem

LogicSystem --> RollService

RollService --> RollResultEvent

RollResultEvent --> EventBus

EventBus --> LogicSystem

Aquí tienes la planificación **día a día** para el **Sprint 1 (Núcleo Jugable)**, en un formato aún más atómico, estructurada en **3 semanas** (15 días laborales). Cada ítem es la unidad mínima de trabajo, con dependencias y duración estimada.

**📅 Semana 1: Inicialización y ECS Básico**

| **Día** | **Tarea** | **Duración** | **Dependencia** |
| --- | --- | --- | --- |
| 1 | **1.1.1** Crear repositorio Git vacío con .gitignore. | 0.25 día | — |
| 1 | **1.1.2** Configurar settings.gradle y build.gradle raíz. | 0.25 día | 1.1.1 |
| 1 | **1.1.3** Añadir submódulos :core y :desktop en settings.gradle. | 0.25 día | 1.1.2 |
| 1 | **1.1.4** Configurar plugin Java y libGDX BOM en core/build.gradle. | 0.25 día | 1.1.3 |
| 2 | **1.2.1** Crear clase DesktopLauncher en desktop/src con LwjglApplication. | 0.25 día | 1.1.4 |
| 2 | **1.2.2** Referenciar GameApp desde DesktopLauncher. | 0.125 día | 1.2.1 |
| 2 | **1.2.3** Ejecutar y verificar “Hello world” (pantalla negra). | 0.125 día | 1.2.2 |
| 3 | **1.3.1** Crear interfaz Component en core.logic.ecs. | 0.166 día | 1.2.3 |
| 3 | **1.3.2** Crear record Entity con UUID id y Map<Class<?>,Component> components. | 0.333 día | 1.3.1 |
| 3 | **1.3.3** Crear interfaz System con método update(float delta). | 0.166 día | 1.3.2 |
| 4 | **1.4.1** Define record PositionComponent(int x,int y). | 0.25 día | 1.3.3 |
| 4 | **1.4.2** Define record RenderComponent(String spritePath). | 0.25 día | 1.4.1 |
| 4 | **1.5.1** Crear clase RenderSystem implementando System. | 0.125 día | 1.4.2 |
| 4 | **1.5.2** Inyectar SpriteBatch en RenderSystem (constructor). | 0.125 día | 1.5.1 |
| 4 | **1.5.3** Implementar loop que recorra entidades con RenderComponent y dibuje rectángulos básicos. | 0.5 día | 1.5.2 |
| 5 | **1.6.1** Crear clase LogicSystem con método vacío update. | 0.25 día | 1.3.3 |
| 5 | **1.6.2** Registrar LogicSystem en bucle principal de GameApp. | 0.25 día | 1.6.1 |
| 5 | **1.7.1** Dibujar líneas verticales de grid cada 32 px en RenderSystem. | 0.25 día | 1.5.3 |
| 5 | **1.7.2** Dibujar líneas horizontales de grid cada 32 px en RenderSystem. | 0.25 día | 1.7.1 |

**📅 Semana 2: AssetManager y Pooling**

| **Día** | **Tarea** | **Duración** | **Dependencia** |
| --- | --- | --- | --- |
| 6 | **1.8.1** Configurar AssetManager en GameApp (instancia única). | 0.125 día | 1.5.3 |
| 6 | **1.8.2** Añadir carga asíncrona de tiles.atlas: assetManager.load(...). | 0.125 día | 1.8.1 |
| 6 | **1.8.3** Llamar a assetManager.finishLoading() antes del primer render. | 0.125 día | 1.8.2 |
| 6 | **1.8.4** Probar que TextureAtlas está disponible en RenderSystem. | 0.125 día | 1.8.3 |
| 7 | **1.9.1** Crear clase EntityPool con métodos borrow() y return(). | 0.25 día | 1.3.2 |
| 7 | **1.9.2** Implementar reutilización de instancias en EntityPool. | 0.25 día | 1.9.1 |
| 7 | **1.9.3** Ajustar RenderSystem y LogicSystem para tomar entidades de EntityPool. | 0.25 día | 1.9.2 |
| 7 | **1.9.4** Pruebas unitarias básicas para EntityPool (借用 y devolución). | 0.25 día | 1.9.3 |

**📅 Semana 3: Consola y Comandos Básicos**

| **Día** | **Tarea** | **Duración** | **Dependencia** |
| --- | --- | --- | --- |
| 8 | **2.1.1** Crear Stage uiStage secundario en GameApp. | 0.125 día | 1.8.4 |
| 8 | **2.1.2** Cargar Skin(Gdx.files.internal("ui/skin.json")). | 0.125 día | 2.1.1 |
| 8 | **2.1.3** Añadir TextField y Table flotante a uiStage. | 0.25 día | 2.1.2 |
| 8 | **2.1.4** Redirigir Gdx.input.setInputProcessor(uiStage). | 0.125 día | 2.1.3 |
| 9 | **2.2.1** Crear clase CommandParser con método parse(String). | 0.125 día | 2.1.4 |
| 9 | **2.2.2** Implementar detección de /roll con expresiones regulares. | 0.125 día | 2.2.1 |
| 9 | **2.2.3** Mapear /spawn y /describe en el mismo parser. | 0.125 día | 2.2.2 |
| 9 | **2.2.4** Probar CommandParser en un test unitario mínimo. | 0.125 día | 2.2.3 |
| 10 | **2.3.1** Registrar listener de TextField que dispare CommandParser. | 0.166 día | 2.1.4 |
| 10 | **2.3.2** Al parsear, emitir eventos a EventBus.fire(...). | 0.166 día | 2.3.1 |
| 10 | **2.3.3** Prueba de integración: escribir en consola y verificar evento. | 0.166 día | 2.3.2 |
| 11 | **2.4.1** En LogicSystem, suscribirse a RollCommandEvent. | 0.125 día | 2.3.3 |
| 11 | **2.4.2** Invocar RollService.lanzar(notation) y publicar RollResultEvent. | 0.125 día | 2.4.1 |
| 11 | **2.5.1** Suscribirse a RollResultEvent en ConsoleSystem. | 0.125 día | 2.4.2 |
| 11 | **2.5.2** Mostrar el resultado en TextField o log de consola. | 0.125 día | 2.5.1 |
| 12 | **2.6.1** /spawn: en LogicSystem, escuchar SpawnCommandEvent. | 0.166 día | 2.2.3 |
| 12 | **2.6.2** Usar EntityPool.borrow(), añadir componentes y registrar en lista activa. | 0.166 día | 2.6.1 |
| 12 | **2.6.3** Verificar visualización inmediata en grid. | 0.166 día | 2.6.2 |
| 12 | **2.6.4** Test unitario: /spawn crea entidad con componentes correctos. | 0.166 día | 2.6.3 |
| 13 | **2.7.1** /describe: en CommandParser, emitir DescribeCommandEvent. | 0.125 día | 2.2.3 |
| 13 | **2.7.2** En ConsoleSystem, suscribirse a DescribeCommandEvent. | 0.125 día | 2.7.1 |
| 13 | **2.7.3** Recuperar entidad por ID/nombre y formatear texto. | 0.125 día | 2.7.2 |
| 13 | **2.7.4** Mostrar componentes en TextField de consola. | 0.125 día | 2.7.3 |
| 14 | **2.8.1** Añadir scroll y límite de líneas en TextField. | 0.25 día | 2.1.3 |
| 14 | **2.8.2** Implementar historial de comandos (flechas arriba/abajo). | 0.25 día | 2.8.1 |
| 15 | **2.9.1** Ajustar layout de Table: semi-transparente en esquina inferior. | 0.25 día | 2.1.3 |
| 15 | **2.10.1** Escribir tests unitarios para CommandParser, RollService, /spawn y /describe. | 0.75 día | 2.2.4,2.6.4,2.7.4 |

**✔️ Resumen de Dependencias Clave**

1. **1.1.1 → 1.1.2 → 1.1.3 → 1.1.4 → 1.2.1 → 1.2.2 → 1.2.3**
2. **1.3.1 → 1.3.2 → 1.3.3 → [1.4.1,1.4.2] → 1.5.1 → 1.5.2 → 1.5.3**
3. **1.6.1 → 1.6.2**
4. **1.7.1 → 1.7.2**
5. **1.8.1 → 1.8.2 → 1.8.3 → 1.8.4**
6. **1.9.1 → 1.9.2 → 1.9.3 → 1.9.4**
7. **2.1.1 → 2.1.2 → 2.1.3 → 2.1.4**
8. **2.2.1 → 2.2.2 → 2.2.3 → 2.2.4**
9. **2.3.1 → 2.3.2 → 2.3.3**
10. **2.4.1 → 2.4.2 → 2.5.1 → 2.5.2**
11. **2.6.1 → 2.6.2 → 2.6.3 → 2.6.4**
12. **2.7.1 → 2.7.2 → 2.7.3 → 2.7.4**
13. **2.8.1 → 2.8.2**
14. **2.9.1**
15. **2.10.1**

Con esta planificación **ultra-atómica**, cada fracción de día tiene una tarea claramente definida y dependencias explícitas. ¡A dar el primer sprint con la máxima precisión!