**Plan de Gestión de Configuraciones**

**Grupo:** CodeTrinity  
**Proyecto:** Piramyd

### **1. Dirección y forma de acceso a la herramienta de control de versiones**

Se utiliza GitHub como herramienta de control de versiones, aprovechando sus funcionalidades de trabajo colaborativo, sistema de ramas y fusión, control de cambios, etiquetado y releases.

* **Repositorio:**<https://github.com/Tuteku/CodeTrinity.git>
* **Acceso:** Vía navegador web o mediante Git CLI (git clone, git pull, git push)

### **2. Dirección y forma de acceso a la herramienta de integración continua (CI)**

Se utilizará GitHub Actions como herramienta de CI/CD. Esta se integra directamente con el repositorio de GitHub, lo cual permite:

* Ejecutar tests automáticos.
* Compilar y empaquetar el juego.
* Generar releases automáticas.

**Acceso:** En la página del repositorio, pestaña "Actions".

### **3. Herramienta de gestión de defectos**

Se empleará **GitHub Issues** para la gestión de defectos, permitiendo:

* Asignación de issues.
* Seguimiento de estados.
* Vinculación con commits y Pull Requests.

**Acceso:** Desde la sección "Issues" del repositorio en GitHub.

### 

### **4. Esquema de directorios**

/

|-- src/ -> Código fuente del juego

|-- assets/ -> Recursos multimedia (imágenes, sonidos, fuentes)

|-- docs/ -> Documentación del proyecto

|-- tests/ -> Pruebas unitarias y de integración

|-- scripts/ -> Scripts de automatización (deploy, setup, etc.)

|-- build/ -> Archivos generados para distribución

### **5. Norma de etiquetado y nombramiento de los archivos**

* **Archivos de código:** PascalCase (ej: MainMenu.java)
* **Documentación:** snake\_case (ej: plan\_configuracion.md)
* **Carpetas:** minúsculas y descriptivas (assets/, scripts/)

### **6. Políticas de fusión y etiquetado según calidad del entregable**

**Flujo de ramas:**

1. Se crea una rama feature (feature/nueva\_funcionalidad).
2. Se fusiona en development y se elimina una vez implementada la feature.
3. Cuando development es estable, se fusiona en release.
4. Se hacen pruebas en release, y si es exitoso, se fusiona en master.
5. En master se etiqueta (tag) la versión.

**Versionado Semántico:**

* vX.Y.Z[-sufijo]
* **X:** Cambios mayores (incompatibles).
* **Y:** Nuevas funcionalidades.
* **Z:** Parches o correcciones menores.
* **Sufijos:** alpha, beta, rc1, etc.

### **7. Forma de entrega de releases**

Se utilizará **GitHub Releases**, permitiendo:

* Publicación de binarios o archivos comprimidos por versión.
* Asociación de releases con tags.
* Descripción de cambios en cada versión.

**Versionado:** Se seguirá la convención vX.Y.Z[-sufijo], automatizado mediante GitHub Actions.

**Instrucciones mínimas de instalación:**

* Descargar el archivo desde la última release publicada.
* Ejecutar el binario o abrir el archivo principal del juego (según plataforma objetivo).

### **8. Herramienta de seguimiento de defectos y su estado**

**Herramienta:** GitHub Issues

**Flujo de trabajo:**

* Cada defecto detectado se registra como issue.
* Se etiqueta según prioridad o tipo (bug, critical, UI, etc.).
* Se asigna a un desarrollador responsable.
* Se vincula a ramas y Pull Requests.
* Se cierra automáticamente al resolver el defecto.

**Acceso:** Desde la página del repositorio, sección "Issues".