Integrantes: Sebastián León, Gregory Salazar

Curso: GR2SW

Fecha: 12 de noviembre de 2025

Proyecto

1. **Definición del Proyecto:**
    - **Nombre del proyecto:** 21A Blackjack
    - **Descripción:**

21A Blackjack es una herramienta interactiva diseñada para asistir a jugadores de Blackjack mediante la recomendación automática de la mejor jugada posible según su mano actual y la carta visible del crupier. Utiliza las reglas estándar de la "Estrategia Básica" y las adapta a tiempo real según el progreso de la partida. Además, incluye un módulo de simulación de apuestas basado en conteo de cartas, generando sugerencias responsables sobre cuándo aumentar o disminuir la apuesta. El sistema está pensado para ejecutarse en navegador o escritorio, manteniendo un diseño modular.

2. **Plan Maestro SCM**

Fase 1: ANTES de Escribir la Primera Línea de Código (Planificación y Diseño)

Herramientas: Git y GitHub.

Repositorio: Será un monorepo llamado BlackJack-EPN. La rama principal se llamará "main" y será protegida para que nadie pueda hacer push directo.

Workflow: Las ramas de trabajo se llamarán "[feature,bugfix]/[nombre-feature,bug]".

Se prohíbe el merge directo; solo se usa Squash Merge o Merge Commit, según el tamaño del PR.

Política de ramas: Se usará feature branching, por cada funcionalidad a implementar existirá una rama feature de desarrollo que luego se deberá integrar a la rama main a través de un Pull Request obligatorio.

Artefactos (No código): Los requisitos estarán en un archivo Markdown dentro de una carpeta /docs en el repositorio. Los diseños (Mockups/UI) estarán en un enlace de excalidraw dentro del README.md del proyecto.

Fase 2: DURANTE el Desarrollo (Codificación y Pruebas)

1) **Gestión de Cambios en Acción**

El Proceso del Pull Request (PR)

Un Pull Request podrá ser aprobado únicamente cuando cumpla con estas condiciones:

- **Funcionalidad completa y probada:** La característica solicitada funciona correctamente y cubre todos los escenarios indicados en el Issue (por ejemplo: sugerir acción óptima según la tabla de estrategia básica).
- **Revisión obligatoria:** Al menos un revisor debe aprobar el PR, revisando calidad del código, claridad y consistencia con el estilo del proyecto.
- **No introduce regresiones:** Todas las pruebas existentes deben continuar pasando.
- **Documentación mínima actualizada:** Si la funcionalidad lo requiere (por ejemplo una nueva recomendación o regla), se debe actualizar el archivo correspondiente dentro de /docs.
- **Sin conflictos:** El PR debe poder fusionarse a main sin conflictos.

Trazabilidad

Para mantener un registro claro entre tareas del backlog y el código:

- Todos los commits deben incluir un ID de Issue.
  - Ej. Añade cálculo de probabilidad de bust con cartas altas (issue #27).
- Los PR deberán estar asociados obligatoriamente a un Issue, ya sea una funcionalidad, bug o tarea técnica.

De esta forma se garantiza trazabilidad completa entre planificación, ejecución y repositorio.

2) Integración Continua (CI)

Para evitar romper el proyecto a medida que se agregan funciones:

- Se implementarán Workflows de GitHub Actions que se ejecutarán en:
  - Cada push a ramas feature/
  - Cada push a ramas bugfix/
  - Cada PR hacia main

Workflows:

Workflow 1 – Pruebas Lógicas del Motor de Blackjack

Ejecuta automáticamente:

- pruebas unitarias del motor de decisiones,
- pruebas del conteo de cartas,
- validaciones de cálculo de apuestas recomendadas.

Si una prueba falla, el merge se bloquea.

Workflow 2 – Linter y Estilo de Código

Valida:

- convenciones de nombres,
- consistencia del formato,
- ausencia de imports innecesarios.

Workflow 3 – Análisis de Seguridad Básico

Revisa dependencias vulnerables o inseguras. Esto asegura que cada cambio mantenga la estabilidad del proyecto.

3) **Gestión de Línea Base (Baselining)**

Para marcar los hitos del desarrollo y establecer puntos de referencia estables, se crearán tags usando versionamiento semántico. Cada tag implica una línea base probada que permite:

- Retroceder fácilmente a una versión estable.
- Realizar pruebas incrementales.
- Documentar avances significativos del proyecto.

Hitos del desarrollo

- **v0.1-alpha**: Motor básico (hit/stand).
- **v0.3-alpha**: Soporte para split, double y estrategia avanzada.
- **v0.5-beta**: Módulo de conteo y sugerencias de apuesta.
- **v1.0-release**: Versión completa para usuarios externos.

Fase 3: DESPUÉS del Lanzamiento (Despliegue y Mantenimiento)

1) **Gestión de releases y despliegue (CD):**

El código de main llega a producción automáticamente usando herramientas de despliegue continuo. El trigger que inicia el despliegue será un merge a la rama main.

El esquema de versionamiento a usar es el semántico: MAJOR.MINOR.PATCH. La manera de incrementar los cambios será: cambios que rompen la compatibilidad, nueva función y arreglo de bus, respectivamente.

2) **Plan de mantenimiento proactivo:**

Mantenimiento correctivo: Para el arreglo de bus, se creará una rama "[hotfix/descripción-hotfix]" desde main, se arreglará el bug, se hará un merge con main y se etiquetará como "X.X.n, donde n representa el número de bugs arreglados".

Mantenimiento adaptativo: Cada 6 meses se revisará la compatibilidad entre sistemas operativos, navegadores y se actualizará APIs de terceros.

Mantenimiento perfectivo: La deuda técnica se reducirá cada última semana del mes a través de refactorizaciones y optimización de consultas a la base de datos.

Mantenimiento preventivo: Para evitar problemas futuros se usarán herramientas para analizar automáticamente dependencia de bibliotecas de Python y vulnerabilidades de seguridad. Además, los pull requests automáticos generados serán revisados cada día lunes.