Backgammon Computación 2025

Resumen:

El objetivo de este documento de desarrollo es guiar el proceso de creación de un juego de Backgammon utilizando el lenguaje de programación Python, siguiendo las reglas y mecánicas establecidas. El juego resultante permitirá a los usuarios jugar partidas de Backgammon tanto en una interfaz de consola como en una interfaz gráfica desarrollada con la biblioteca Pygame.

Jugar online:

https://www.ludoteka.com/clasika/backgammon-es.html

1. Introducción:

Este documento detalla el proceso de desarrollo de un juego de Backgammon en Python. El juego se implementará inicialmente en una interfaz de línea de comando, y también contará con una interfaz gráfica usando la biblioteca Pygame. El diseño del juego separará completamente la lógica de negocio de las capas de presentación.

2. Requisitos del Juego:

El juego de Backgammon a implementar debe cumplir con los siguientes requisitos:

2.1. Tablero de juego

El tablero debe contener 24 triángulos (llamados puntos), divididos en cuatro cuadrantes. Cada jugador comienza con 15 fichas en posiciones estándar.

2.2. Fichas y dados

Cada jugador debe tener 15 fichas. Se utilizarán dos dados de seis caras para determinar los movimientos. Las tiradas dobles permiten repetir los valores.

2.3. Jugabilidad

Los jugadores deben mover sus fichas siguiendo las reglas tradicionales del juego, incluyendo movimientos válidos, capturas y reingresos desde la barra. Se debe implementar la condición de victoria.

3. Diseño del Juego:

3.1. Lógica Central (core/)

El desarrollo incluirá clases separadas para Game, Board, Dice y Player, que representarán la lógica del juego de forma independiente de la interfaz de usuario.

3.5.1. Línea de Comando (Obligatorio):

La modalidad de línea de comando permitirá a los jugadores interactuar con el juego a través de comandos de texto en la consola. Los jugadores podrán colocar piezas, realizar intercambios y llevar un seguimiento de la puntuación mediante la entrada y salida de texto. Esta modalidad es obligatoria para garantizar que el juego sea accesible y utilizable en entornos que no admitan interfaces gráficas.

3.5.2. Gráficos con Pygame (Obligatorio):

Además de la modalidad de línea de comando, se brindará la opción de jugar el juego en una interfaz gráfica utilizando la biblioteca Pygame (ver https://www.pygame.org/docs/). Esta modalidad proporcionará una experiencia visual y atractiva para los jugadores, con representaciones gráficas del tablero, las fichas y las acciones del juego. Los jugadores podrán interactuar con la interfaz gráfica utilizando el ratón y el teclado.

Es importante destacar que la elección de utilizar la modalidad de gráficos con Pygame es opcional y no afectará la funcionalidad subyacente del juego. El diseño del juego está cuidadosamente estructurado para garantizar que las mecánicas y reglas del juego se mantengan consistentes en ambas modalidades de interfaz.

La capacidad de jugar tanto desde la línea de comando como con gráficos opcionales brinda flexibilidad a los jugadores para seleccionar la modalidad que mejor se adapte a sus preferencias y entorno de juego. El objetivo es proporcionar una experiencia del juego completa y agradable, sin importar la elección de interfaz.

3.6. Guardado de Partidas en Redis (0pcional)

Para mejorar la experiencia del juego y brindar a los jugadores la posibilidad de retomar partidas en curso, se implementará un sistema de guardado de partidas utilizando la base de datos en memoria Redis. El guardado de partidas permitirá a los jugadores cerrar el juego y reanudarlo más tarde desde el punto exacto donde lo dejaron.

3.4. Estructura del Proyecto

```
/backgammon/
├── core/            → lógica del juego
├── cli/             → CLI
├── pygame_ui/       → interfaz gráfica
├── assets/          → imágenes, sonidos

├── tests/          → tests
└── requirements.txt
```

4. Desarrollo y Buenas Prácticas

4.1. Cobertura del Código

Se deberá alcanzar al menos 90% de cobertura con pruebas unitarias para la lógica central del juego.

Code Climate??

4.2. Desarrollo Incremental

Se realizarán avances quincenales en el proyecto, respetando entregas y commits distribuidos en el tiempo.

4.3. Documentación y Calidad

El proyecto deberá incluir docstrings, README.md y un CHANGELOG.md. Se recomienda seguir los principios SOLID.

4.3. Integración Continua y Control de Calidad:

Se establecerá un sistema de integración continua utilizando herramientas como CircleCI o Travis. Cada commit desencadenará pruebas unitarias automatizadas para verificar la funcionalidad existente. Además, se utilizará CodeClimate o coveralls.io para medir y rastrear el porcentaje de cobertura del código en tiempo real, asegurando que el equipo esté al tanto de la calidad del código en todo momento.

4.4. Inclusión de Prompts

Todos los alumnos deberán incluir, en el repositorio, archivos con los **prompts exactos** que utilizaron para el desarrollo, testing y documentación cuando hayan recurrido a herramientas de IA (por ejemplo: chatbots, generadores de código, LLMs, etc.).

Archivos obligatorios:

- **prompts-desarrollo.md**
- **prompts-testing.md**
- **prompts-documentacion.md**

Por cada prompt debe quedar registrado:

- Modelo / herramienta usada (nombre y versión si corresponde).
- El texto exacto del prompt (sin reescrituras subjetivas).
- Instrucciones del sistema (si las hubo).
- Respuesta/resultado completo devuelto por la IA.
- Indicar si la salida fue usada sin cambios, usada con modificaciones (mostrar las modificaciones) o descartada.
- Referencia a los archivos finales que incorporaron contenido generado por IA (ej: core/board.py).

4.5. Justificación escrita (Markdown)

Se deberá agregar un archivo de justificación general en formato Markdown que acompañe al proyecto y que sirva como base para la exposición oral. Requisitos:

- Archivo obligatorio: **JUSTIFICACION.md** (Solo se aceptaran archivos en formato markdown).
- Contenido mínimo:
    - Resumen del diseño general.
    - Justificación de las clases elegidas (por qué, responsabilidades).
    - Justificación de atributos (por qué se eligieron).
    - Decisiones de diseño relevantes
    - Excepciones y manejo de errores (qué excepciones definidas y por qué).
    - Estrategias de testing y cobertura (qué se probó y por qué).
    - Referencias a requisitos SOLID y cómo se cumplen.
    - Anexos: diagramas UML(ej: diagrama de clases).

Esta justificación debe irse actualizando a lo largo del cursado (versionado en git) para mostrar la evolución de las decisiones de diseño.

4.6. Justificación oral

Además de la justificación escrita, incluirá una **instancia de examen oral** cuya finalidad es verificar la comprensión del alumno sobre el proyecto y las decisiones de diseño.
En dicha instancia se abarcaran los siguientes puntos:

- **Exposición general del proyecto actual**: flujo de ejecución, responsabilidades de las clases, puntos clave del diseño y dependencias relevantes entre módulos.
- **Decisiones de diseño**: explicación de las decisiones arquitectónicas importantes (por ejemplo: separación core / UI, elección de estructuras de datos, manejo de errores y excepciones propias), las alternativas consideradas y por qué se descartaron.
- **Estrategia de testing y evidencia**: qué se probó, porqué se eligieron esos tests, y demostraciones ejecutables (via CLI o tests unitarios) que muestren el cumplimiento de requisitos clave.
- **Implementación concreta**: preguntas y demostraciones sobre cualquier parte implementada en el código (por ejemplo: la lógica de movimientos, guardado/recupero de partidas, integración con Redis si se implementó, aspectos de la UI con Pygame, etc.).
- **Evaluación de la comprensión**: la evaluación oral puede incluir preguntas técnicas o conceptuales sobre **cualquier tema visto en la materia** y sobre **cualquier cuestión implementada en el código** del proyecto.
- El objetivo de todo esto es que el alumno demuestre comprensión de la lógica (no sólo que el juego funcione).

5. Diseño Básico Propuesto

5.1. Clases

BackgammonGame → Coordina flujo general
Board → Representa el tablero y puntos
Player → Representa a un jugador
Dice → Lógica de tiradas
Checker → Representa cada ficha
CLI → Interfaz de texto
PygameUI → Interfaz gráfica

5.2. Relaciones

BackgammonGame
├─ Board
├─ Player
├─ Dice
└─ Interfaces: CLI / PygameUI

6. Aprobación de la materia

Para alcanzar la aprobación de la materia el único requisito es el cumplimiento de todas las features establecidas en puntos anteriores en una fecha previa al término del segundo semestre de acuerdo a las siguientes consideraciones:

Los sprints establecidos para las presentaciones intermedias tendrán un plazo de 14 días

Cada sprint deberá contener al menos 10 commits de fecha diferente en el repositorio oficial entregado por la asignatura, no se permitirá el uso de repositorio personal. En el caso de no alcanzar el requisito establecido, el repositorio deberá contar con al menos 6 commits acumulando los 4 faltantes para el siguiente sprint, la cantidad de faltantes nunca podrá exceder en ningún caso el valor de 4. La falla en este requisito implica quedar fuera del cursado de la materia.

Debe agregarse obligatoriamente al repositorio un archivo CHANGELOG.md de acuerdo a los criterios recomendados en https://keepachangelog.com/en/1.1.0/ en el que se podrá apreciar las tareas realizadas en cada sprint. La falla en este requisito implica quedar fuera del cursado de la materia.

El desarrollo del trabajo debe cumplir los principios SOLID que garantizan la aplicación correcta del paradigma orientado a objetos. No existe justificación para dejar partes no orientadas a objeto dentro del desarrollo del trabajo.

Todos los atributos de todas las clases deben contener como prefijo y postfijo los símbolos "__". Ejemplo: el atributo velocidad debe quedar como __velocidad__ . Asimismo, el único lugar viable para encontrar un atributo en el código es junto a la palabra self.

La aplicación debe incluir documentación del "estilo" Docstrings donde se indique por cada función la información relevante, mínimo lo que la función recibe, lo que la función hace y lo que la función devuelve.

https://realpython.com/documenting-python-code/

El repositorio contendrá un archivo README.md con una explicación detallada de cómo debe ponerse en funcionamiento el Backgammon para modo testing y para modo juego desplegados ambos con Docker.

El trabajo será desarrollado exclusivamente en forma individual. La falla en este requisito implica quedar fuera del cursado de la materia.

Cada commit debe reflejar una evolución del proyecto hacia los objetivos establecidos. La variación porcentual entre un commit y el siguiente será decreciente a medida que se acerca el final del trabajo, entendiendo que no pueden haber grandes cambios hacia el final salvo que se haya iniciado el desarrollo nuevamente, en cuyo caso primero debe ser consensuado con los profesores de la asignatura.

La versión "entregable" de la aplicación DEBE estar en la branch MAIN del repositorio; pueden crearse las ramas que cada alumno considere necesaria para su trabajo, pero el producto final estará en la branch MAIN. Debe existir obligatoriamente una regla de protección que impida hacer push directos sobre la rama MAIN.

La calificación de CodeClimate debe ser A al momento de la entrega del trabajo permitiendo 0 issues al final del trabajo, todos los issues deben ser salvados sin excepción.

Las frases “no sabía”, “no me dijeron”, “no me enteré”, “no estaba al tanto”, “mis compañeros no sabían”, etc, etc, pueden usarse pero no tienen ninguna validez en el contexto de lo que debe cumplirse para aprobar la materia.

6.1. Sospecha de copia y trazabilidad

En caso de sospecha fundada de copia entre repositorios o proyectos, es decir, cuando se detecte similitud notable en código, estructura, commits o artefactos **se considerará que el repositorio cuyo último commit tenga fecha/hora posterior es el repositorio que copió**. La comparación se realiza tomando los metadatos del commit (fecha y hora explicitadas en el log de git) y la evidencia técnica.

Para facilitar la investigación, todo alumno debe conservar y mantener:

- Historial de commits completo en el repositorio oficial, es decir, una vez que un commit se sube a **main**, no se permite hacer acciones que modifiquen, eliminen o sustituyan esos commits en el registro histórico de la rama (fechas, hashes, etc.).
- **prompts-desarrollo.md, prompts-testing.md, prompts-documentacion.md** y **JUSTIFICACION.md** actualizados.

En caso de que el alumno no pueda aportar trazabilidad suficiente para demostrar autoría, se aplicarán las sanciones que determine el equipo docente.

7. Referencias

https://es.wikipedia.org/wiki/Backgammon

https://www.pygame.org/docs/

https://realpython.com/documenting-python-code/

https://keepachangelog.com/en/1.1.0/