Trabajo Práctico

Fecha de Entrega: *18 de Septiembre*

Introducción

PUBG es un juego de disparos en primera y tercera persona estilo Battle Royale que enfrenta a más de 90 jugadores en una gran isla donde los equipos y los jugadores luchan a muerte hasta que queda uno.

El sitio pubg.op.gg publica estadísticas sobre este juego.

Queremos analizar un dataset con dumps de este sitio para mejorar las estrategias de juego.

Objetivo

Implementar una aplicación en Rust para procesamiento de información, aprovechando las ventajas del modelo Fork-Join, utilizando el dataset https://www.kaggle.com/datasets/skihikingkevin/pubg-match-deaths

La información a obtener del mismo incluye:

- El top 10 de armas (campo `killed_by`) que produjeron la mayor cantidad de muertes. En caso de empate, resolver alfabeticamente.
  - De cada arma, el % del total de muertes que produjo, redondeado a dos decimales.
  - El promedio de distancia entre el asesino y la víctima para esa arma (`((killer_position_x - victim_position_x).powi(2) + (killer_position_y - victim_position_y).powi(2)).sqrt();`, redondeado a dos decimales.
-
- EL top 10 de jugadores (campo `killer_name`) que produjeron la mayor cantidad de muertes. En caso de empate, resolver alfabeticamente.
  - De cada jugador, el total de muertes que produjo.
  - El top 3 de armas mas utilizadas por el jugador, junto con su % de uso respecto del total de muertes ocasionadas por ese jugador, redondeado a dos decimales. En caso de empate, resolver alfabeticamente.

Requerimientos

- La aplicación debe recibir como parámetro de linea de comandos el path a un directorio, y debe procesar todos los archivos `.csv` en el mismo. Los archivos a procesar corresponden con el formato del directorio `deaths` dentro del dataset.
- Debe recibir un segundo parámetro entero por linea de comandos indicando la cantidad de worker threads con la cual procesar la información
- Debe recibir un tercer parámetro con el nombre del archivo de salida como resultado del procesamiento.

En resumen, la aplicación será ejecutada como `cargo run <input-path> <num-threads> <output-file-name>`

- El formato del archivo de salida debe ser

```
{
    "padron": <número de padron del alumno>,
    "top_killers": {
        "<killer_name 1>": {
            "deaths": <cantidad total de muertes ejecutadas por el jugador>,
            "weapons_percentage": {
                "<weapon name 1>": <% de uso respecto del total de muertes
ocasionadas por ese jugador>,
                ...
            }
        },
        ...
        "<killer_name N>" : {
            ...
        }
    },
    "top_weapons": {
        "<killed_by 1>": {
          "deaths_percentage": <% del total de muertes que produjo el arma>,
          "average_distance": <promedio de distancia entre el asesino y la
víctima para el arma>
        },
        ...
        "<killed_by N>": {

        },
    }
}
```

Requerimientos no funcionales

Los siguientes son los requerimientos no funcionales para la resolución de los ejercicios:

- El proyecto deberá ser desarrollado en lenguaje Rust, usando las herramientas de la biblioteca estándar.
- El archivo Cargo.toml se debe encontrar en la raíz del repositorio, para poder ejecutar correctamente los tests automatizados
- Se deberán utilizar las herramientas de concurrencia correspondientes al modelo forkjoin
- No se permite utilizar **crates** externos, salvo los explícitamente mencionados en este enunciado, en los ejemplos de la materia, o autorizados expresamente por los profesores. Para el procesamiento de JSON se puede utilizar el crate `serde_json`.
- El código fuente debe compilarse en la última versión stable del compilador y no se permite utilizar bloques unsafe.
- El código deberá funcionar en ambiente Unix / Linux.
- El programa deberá ejecutarse en la línea de comandos.
- La compilación no debe arrojar **warnings** del compilador, ni del linter **clippy**.
- Las funciones y los tipos de datos (**struct**) deben estar documentadas siguiendo el estándar de **cargo doc**.
- El código debe formatearse utilizando **cargo fmt**.
- Cada tipo de dato implementado debe ser colocado en una unidad de compilación (archivo fuente) independiente.

Entrega

La resolución del presente proyecto es individual.

La entrega del proyecto se realizará mediante Github Classroom. Cada estudiante tendrá un repositorio disponible para hacer diferentes commits con el objetivo de resolver el problema propuesto. Se recomienda iniciar tempranamente y hacer commits pequeños agreguen funcionalidad incrementalmente. Se podrán hacer commit hasta el día de la entrega a las 19 hs Arg, luego el sistema automáticamente quitará el acceso de escritura.

Evaluación

Principios teóricos y corrección de bugs

La evaluación se realizará sobre Github, pudiendo el profesor hacer comentarios en el repositorio y solicitar cambios o mejoras cuando lo encuentre oportuno, especialmente debido al uso incorrecto de herramientas de concurrencia.

Casos de prueba

Se someterá a la aplicación a diferentes casos de prueba que validen la correcta aplicación de las herramientas de concurrencia, por ejemplo, la ausencia de deadlocks.

Además la aplicación deberá respetar los formatos de salida y valores esperados de los resultados, y deberá mostrar algún incremento en performance cuando la ejecución de la misma se hace con varios hilos en un ambiente multiprocesador.

Organización del código

El código debe organizarse respetando los criterios de buen diseño y en particular aprovechando las herramientas recomendadas por Rust. Se prohibe el uso de bloques `unsafe`.

Tests automatizados

La presencia de tests automatizados que prueben diferentes casos, en especial sobre el uso de las herramientas de concurrencia es un plus.

Presentación en término

El trabajo deberá entregarse para la fecha estipulada. La presentación fuera de término sin coordinación con antelación con el profesor influye negativamente en la nota final.