
Riot API

# Jungle Path Challenge

Italo Guasti

Ouro Preto
17 de fevereiro de 2025

# Sumário

# 1 Introdução

Este relatório apresenta a implementação desenvolvida utilizando a **Riot API**. O código consiste em acessar a API da Riot Games para selecionar e analisar um jogo de SoloQ, focando no mapeamento do "Jungle Pathing"de um jogador específico que atuou como jungler.

O objetivo principal foi construir um histórico detalhado dos locais visitados pelo jungler durante a partida, além de coletar informações relevantes como kills, mortes, ouro acumulado e outros dados que pudessem fornecer uma visão abrangente do desempenho e das decisões estratégicas tomadas ao longo do jogo. A solução foi implementada em Python, utilizando diversas bibliotecas para realizar a integração com a API, processamento de dados e visualização dos resultados.

# 2 Desenvolvimento

Detalhes referentes a construção do projeto.

## 2.1 Linguagem Utilizada

A linguagem de programação escolhida para este desafio por mim foi o **Python**. Python foi selecionado por ser uma linguagem amplamente utilizada no desenvolvimento de software, especialmente em tarefas que envolvem integração com APIs, processamento de dados e visualização de informações. Sua sintaxe clara e concisa permite um desenvolvimento rápido e eficiente, além de contar com uma vasta quantidade de bibliotecas que facilitam a manipulação de dados e a criação de gráficos. Além disso, Python é amplamente adotado na indústria de tecnologia e é conhecido por sua versatilidade, o que o torna uma escolha ideal para este tipo de desafio técnico. .

## 2.2 Bibliotecas Utilizadas

Para a implementação da solução proposta, foram utilizadas diversas bibliotecas Python, cada uma desempenhando um papel crucial no desenvolvimento do projeto:

- **requests**: Essa biblioteca foi utilizada para realizar as requisições HTTP à API da Riot Games. Ela simplifica o processo de comunicação com a API, permitindo enviar solicitações e receber respostas de forma eficiente e segura.
- **pandas**: Utilizada para manipulação e análise de dados. A capacidade do `pandas` de trabalhar com grandes volumes de dados tabulares facilitou a organização e filtragem das informações obtidas da API.
- **matplotlib**: Esta biblioteca foi escolhida para a visualização dos dados. Com o `matplotlib`, o objetivo foi criar um gráfico que ilustrava o movimento do jungler no mapa.
- **dotenv**: Utilizada para gerenciar as variáveis de ambiente, especificamente a chave de API da Riot Games. O uso de `.env` permite que informações sensíveis, como a chave da API, sejam armazenadas de forma segura, sem a necessidade de hardcoding no código-fonte.

## 2.3 A API da Riot


A **API da Riot Games** é uma ferramenta poderosa que permite o acesso a uma vasta quantidade de dados relacionados aos jogos, jogadores e estatísticas da plataforma. Os principais endpoints da API utilizados neste projeto foram:

**Obtenção do PUUID do Jogador**: Para identificar o jogador na partida, foi necessário obter o PUUID (Player Universally Unique Identifier) a partir do nome de invocador e tag line do jogador. Veja o endpoint utilizado abaixo.

PUUID do Jogador

/riot/account/v1/accounts/by-riot-id/gameName/tagLine

**Obtenção dos IDs de Partidas Recentes**: Com o PUUID em mãos, o próximo passo foi obter os IDs das partidas recentes jogadas pelo jogador.

IDs das Partidas

/lol/match/v5/matches/by-puuid/puuid/ids

**Obtenção dos Detalhes de uma Partida Específica**: Com o ID de uma partida selecionada, foi possível acessar informações detalhadas sobre essa partida, incluindo dados específicos dos participantes.

Detalhes da Partida

/lol/match/v5/matches/matchId

**Obtenção da Linha do Tempo de uma Partida**: Para mapear o movimento do jungler no jogo, foi necessário acessar a linha do tempo da partida, que inclui eventos detalhados como compras de itens e posições no mapa.

Linha do Tempo da partida

/lol/match/v5/matches/matchId/timeline

## 2.4 Análise de Dados da Partida

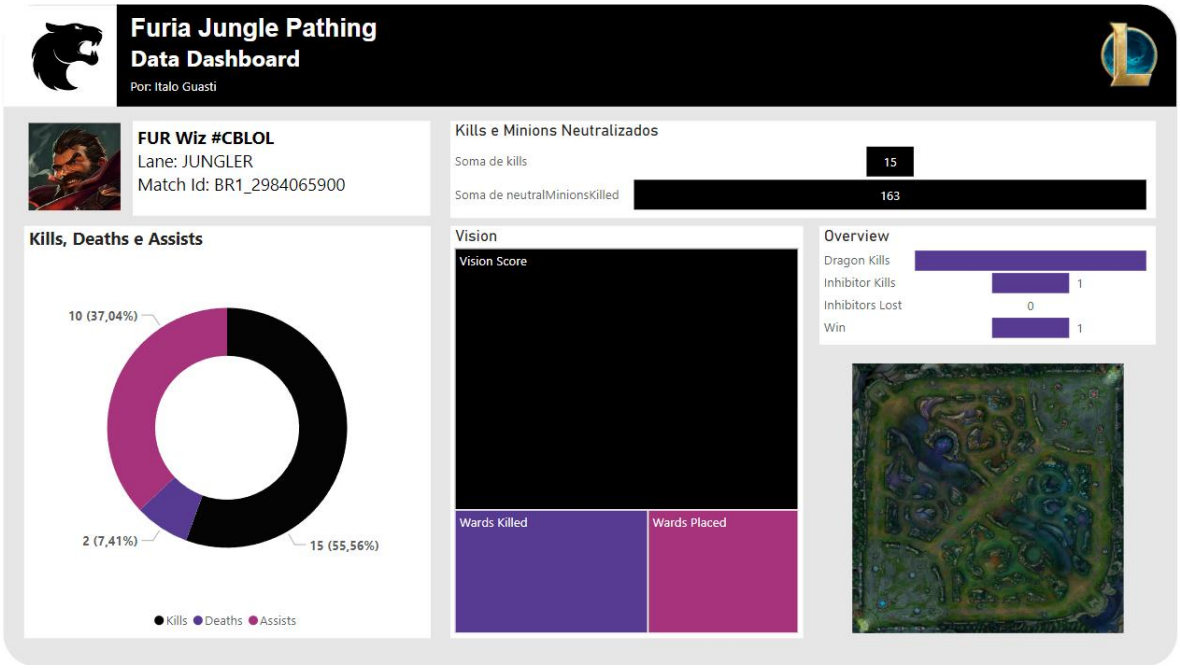
Nesta seção, são analisados os dados detalhados da partida jogada pelo player **'FUR Wiz #CBLOL'** na posição de jungler. O jogador utilizou o campeão **Graves** na função de **JUNGLE**. A análise a seguir apresenta uma visão geral do desempenho do jogador, com base nas estatísticas extraídas da API da Riot Games.


```
{
    "championName": "Graves",
    "teamPosition": "JUNGLE",
    "individualPosition": "JUNGLE",
    "firstBloodKill": "False",
    "kills": 15,
    "deaths": 2,
    "assists": 10,
    "items": [
        3142,
        6676,
        2055,
        6673,
        3036,
        3111,
        3364
    ],
    "neutralMinionsKilled": 163,
    "totalMinionsKilled": 60,
    "dragonKills": 3,
    "baronKills": 0,
    "wardsPlaced": 7,
    "wardKilled": 9,
    "visionScore": 34,
    "inhibitorKills": 1,
    "inhibitorsLost": 0,
    "goldEarned": 15686,
    "win": "True"
}
```


- **Kills e Mortes**: O jogador conseguiu **15 kills** e sofreu apenas **2 mortes**, resultando em uma excelente relação kills/mortes. Combinando isso com **10 assistências**, o jogador teve uma alta participação em combate.
- **Objetivos Neutros**: O jogador teve uma performance significativa na selva, abatendo **163 neutral minions**, o que mostra um bom controle dos recursos na selva. Além disso, foram realizados **3 abates de dragão**, embora nenhum barão tenha sido abatido.
- **Contribuição ao Time**: Colocando **7 wards** e destruindo **9 wards inimigas**, o jogador contribuiu de forma importante para o controle de visão do mapa, refletido em um **vision score** de **34**. O jogador também foi responsável por destruir **1 inibidor inimigo**, sem perder nenhum inibidor em sua própria base.

- **Itens Construídos**: Os itens comprados mostram um foco em dano e resistência, o que é comum para o estilo de jogo do campeão **Graves** na função de jungler. A combinação de itens indica um equilíbrio entre dano crítico e sobrevivência, permitindo ao jogador ser eficaz tanto em lutas quanto na selva.
- **Ouro e Vitória**: O jogador acumulou **15.686 de ouro** ao longo da partida, o que foi convertido em uma build poderosa que contribuiu significativamente para a vitória do time.

É possível visualizar os dados também no dashboard abaixo, assim como este relatório ele também está disponível no Github.




## 2.5 O Código

Descrição detalhada do funcionamento do código, desde o início até a geração da visualização final.

**1. Carregamento das Configurações e Inicialização**
O código começa carregando as configurações necessárias para o funcionamento, incluindo a chave da API da Riot Games, armazenada de forma segura em um arquivo **.env**.

**2. Obtenção do PUUID do Jogador**
Com a chave da API em mãos, o próximo passo é obter o PUUID (Player Universally Unique Identifier) do jogador de interesse. Para isso, o código faz uma requisição ao endpoint **/riot/account/v1/accounts/by-riot-id/gameName/tagLine**, utilizando o nome do jogador e sua tag line. O PUUID é um identificador único que será utilizado em todas as chamadas subsequentes à API para identificar o jogador.

**3. Recuperação dos IDs de Partidas Recentes**
Após obter o PUUID, o código faz uma nova requisição à API, desta vez ao endpoint **/lol/match/v5/matches/by-puuid/puuid/ids**, para recuperar uma lista de IDs das partidas mais recentes jogadas pelo jogador. Dentre essas partidas, uma específica é selecionada manualmente para análise detalhada.

**4. Obtenção dos Detalhes da Partida**
Com o ID da partida selecionada, o código faz uma requisição ao endpoint **/lol/match/v5/matches/matchId** para obter todos os detalhes dessa partida. Os dados retornados incluem informações sobre o modo de jogo, a duração da partida, o timestamp de início e fim, além de detalhes individuais sobre cada participante, como o campeão jogado, número de kills, mortes, assistências, quantidade de ouro acumulado, e muito mais.

**5. Processamento dos Dados do Jogador**
Com os dados da partida em mãos, o código filtra as informações para focar apenas no jogador identificado pelo PUUID. Aqui, são extraídas informações específicas sobre o desempenho do jogador, como o número de neutral minions mortos, dragões abatidos, torres destruídas, e outras estatísticas relevantes que ajudam a entender a performance do jungler durante a partida.

**6. Obtenção da Linha do Tempo da Partida**
Para mapear o movimento do jungler no mapa durante a partida, o código faz uma requisição ao endpoint **/lol/match/v5/matches/matchId/timeline**, que retorna a linha do tempo da partida. Esta linha do tempo contém eventos detalhados, incluindo a posição do jogador em diferentes momentos do jogo, compras de itens e abates de campeões.

**7. Extração do Jungle Pathing**
A partir dos dados da linha do tempo, o código filtra os eventos relevantes que incluem a posição do jogador no mapa. Esses eventos são organizados em ordem cronológica para criar um histórico detalhado dos locais visitados pelo jungler ao longo do jogo. As coodernadas de posição **(x e y)** são extraídas e armazenadas para visualização posterior.

**8. Visualização do Jungle Pathing**
Finalmente, o código utiliza a biblioteca **matplotlib** para gerar uma visualização gráfica do jungle pathing. O gráfico resultante mostra o caminho percorrido pelo jungler no mapa, com indicações de eventos importantes como compras de itens e abates. Essa visualização proporciona uma visão clara e detalhada das movimentações estratégicas do jungler ao longo da partida. **No entanto, este objetivo não foi concluido com sucesso**.

## 2.6 Resultados

O desenvolvimento deste desafio técnico trouxe resultados bastante positivos e proporcionou uma experiência enriquecedora na integração com a API da Riot Games. A partir dos dados obtidos, foi possível realizar uma análise detalhada do desempenho do jungler, demonstrando habilidades técnicas em extração e manipulação de dados, bem como na geração de insights valiosos sobre a partida analisada.

**Pontos Positivos**

1. **Integração com a API da Riot Games**: A comunicação com a API foi realizada de forma eficiente, permitindo a extração de informações detalhadas sobre o jogador e a partida. A escolha dos endpoints foi precisa e atendeu às necessidades do desafio, permitindo a obtenção de dados cruciais para a análise.

2. **Processamento dos Dados**: O código foi capaz de processar as informações do jogador, extraindo e organizando dados sobre kills, mortes, assistências, e desempenho em objetivos neutros. Esses dados foram fundamentais para construir uma análise sólida do impacto do jungler na partida.

3. **Análise Detalhada do Desempenho**: Com base nos dados processados, foi possível realizar uma análise detalhada do desempenho do jungler, destacando as contribuições do jogador para a vitória do time. Essa análise foi apresentada de maneira clara e estruturada, permitindo uma compreensão profunda do papel do jungler na partida.

**Desafios e Limitações** Embora grande parte do desafio tenha sido completada com sucesso, houve uma dificuldade específica na etapa **8. Visualização do Jungle Pathing** . Embora o código tenha gerado um gráfico, os valores das coordenadas extraídas da API foram muito baixos, o que resultou em uma visualização inadequada da trajetória do jungler pelo mapa. A trajetória esperada não foi corretamente exibida, o que sugere que os valores de posição obtidos da API podem não ser os corretos.

Essa limitação pode estar relacionada à complexidade do body do endpoint de timeline, que é bastante extenso e detalhado. É possível que a extração dos dados de posição necessite de uma abordagem diferente para garantir a precisão das coordenadas e, consequentemente, uma visualização correta do jungle pathing.

## 2.7 Conclusão

Apesar da limitação encontrada na visualização do jungle pathing, o desafio foi uma oportunidade valiosa para demonstrar habilidades técnicas e aprender mais sobre a integração com a API da Riot Games. O processo de extração, processamento e análise dos dados foi realizado com sucesso, gerando resultados significativos e demonstrando uma capacidade analítica sólida.