Otimização de um Algoritmo de Grafo

Gustavo Rubia RA:17.01736-0

Marcello Beer RA:17.00865-4

Rodrigo Carvalho RA: 16.03378-7

Lucas Coelho RA:15.03863-7

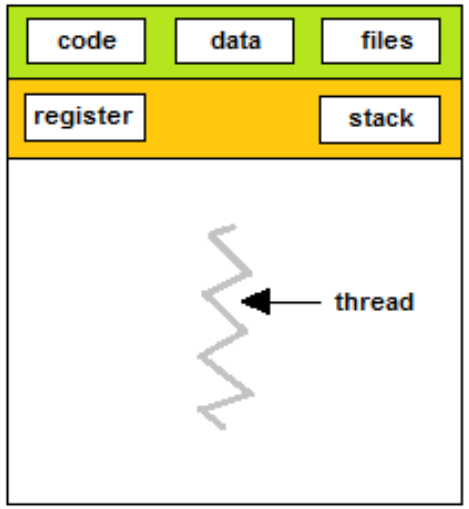
Guilherme Tabacow RA:17.00666-0

Raphael Hungria RA:17.04015-9

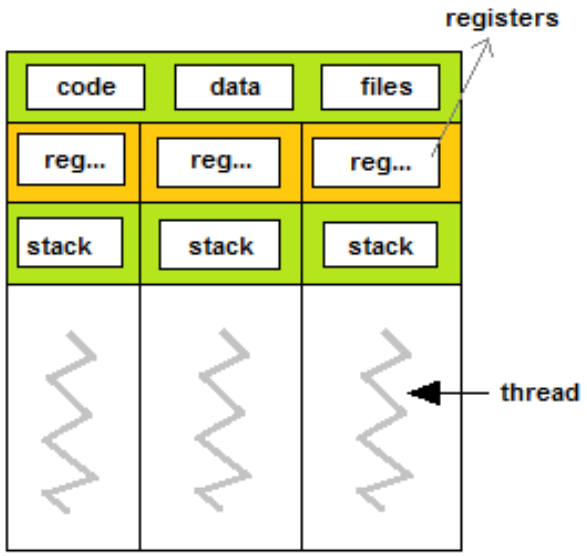
Agenda

- Single Thread vs. Multi Threads
- Grafos
- Algoritmo de Dijkstra
- Algoritmo A*
- Overview do Programa Utilizado
- Análise de Desempenho (Single Thread)
- Análise de Desempenho (Multi Thread)
- Conclusão

Single Thread vs. Multi Threads

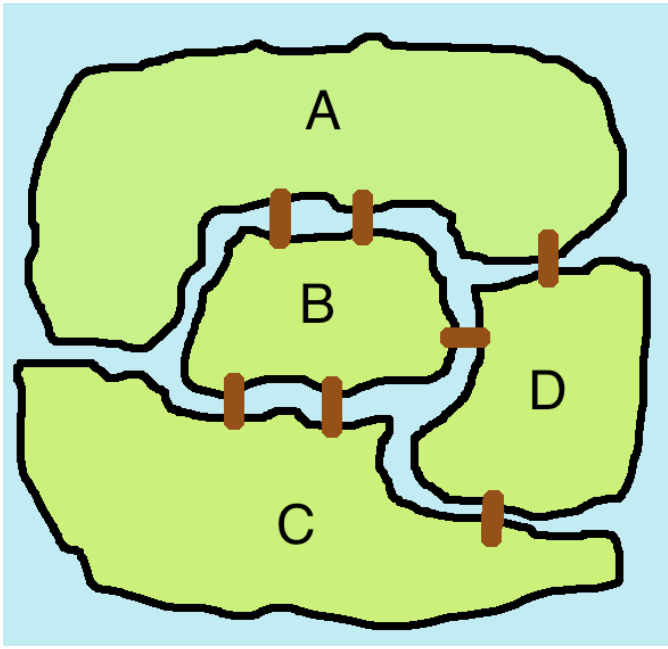


single-threaded process



multithreaded process

Surgimento da Teoria dos Grafos



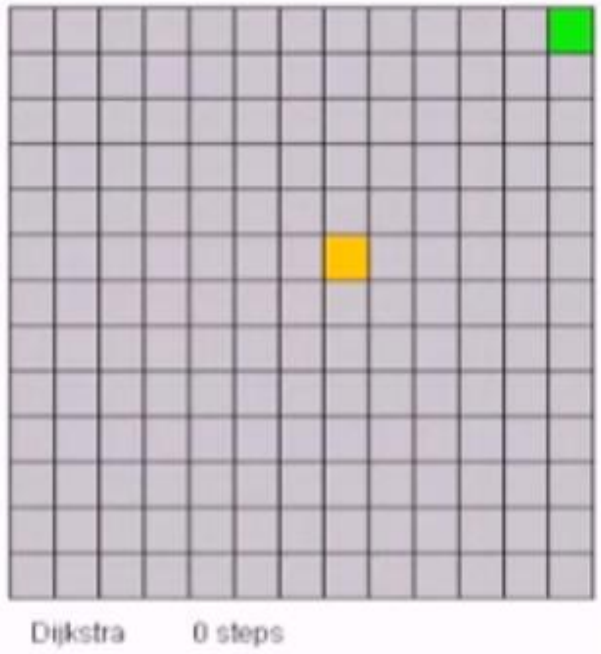
Tipos de Grafos e Aplicações

Algoritmo de Dijkstra

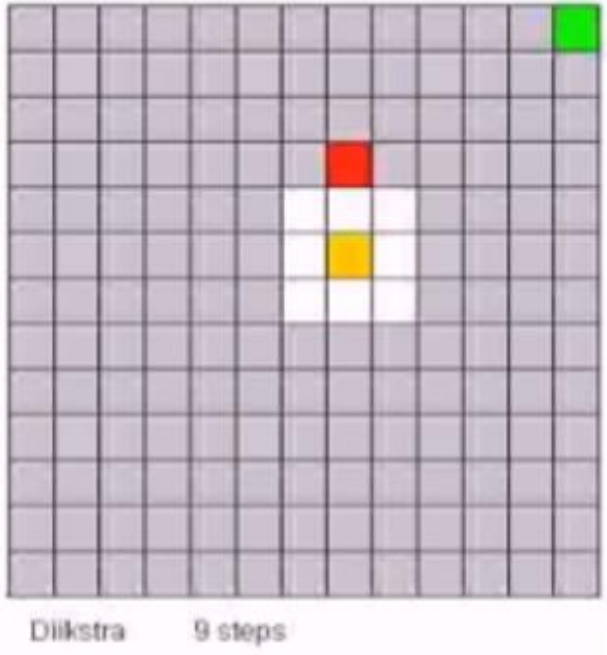
- Criado por Edsger Dijkstra em 1959
- Utilizado em path finding
- Encontra o caminho mais curto
- Analisa **todas** as opções mais curtas

Algoritmo em prática

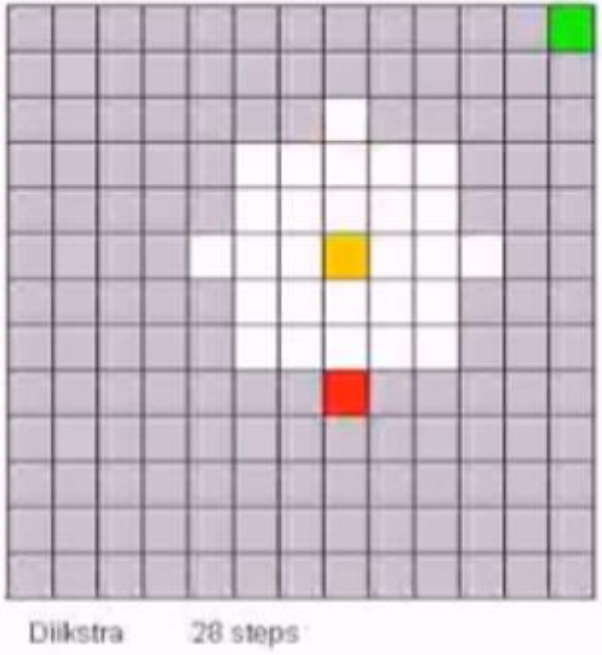
0 iterações



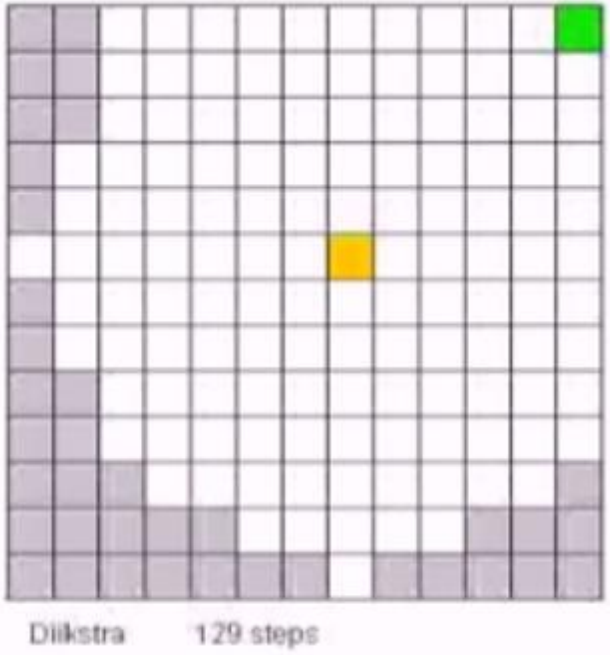
9 iterações



28 iterações



129 iterações



Não explorado
Em análise
Explorado
Obstáculo
Ponto de partida
Destino

Algoritmo A*

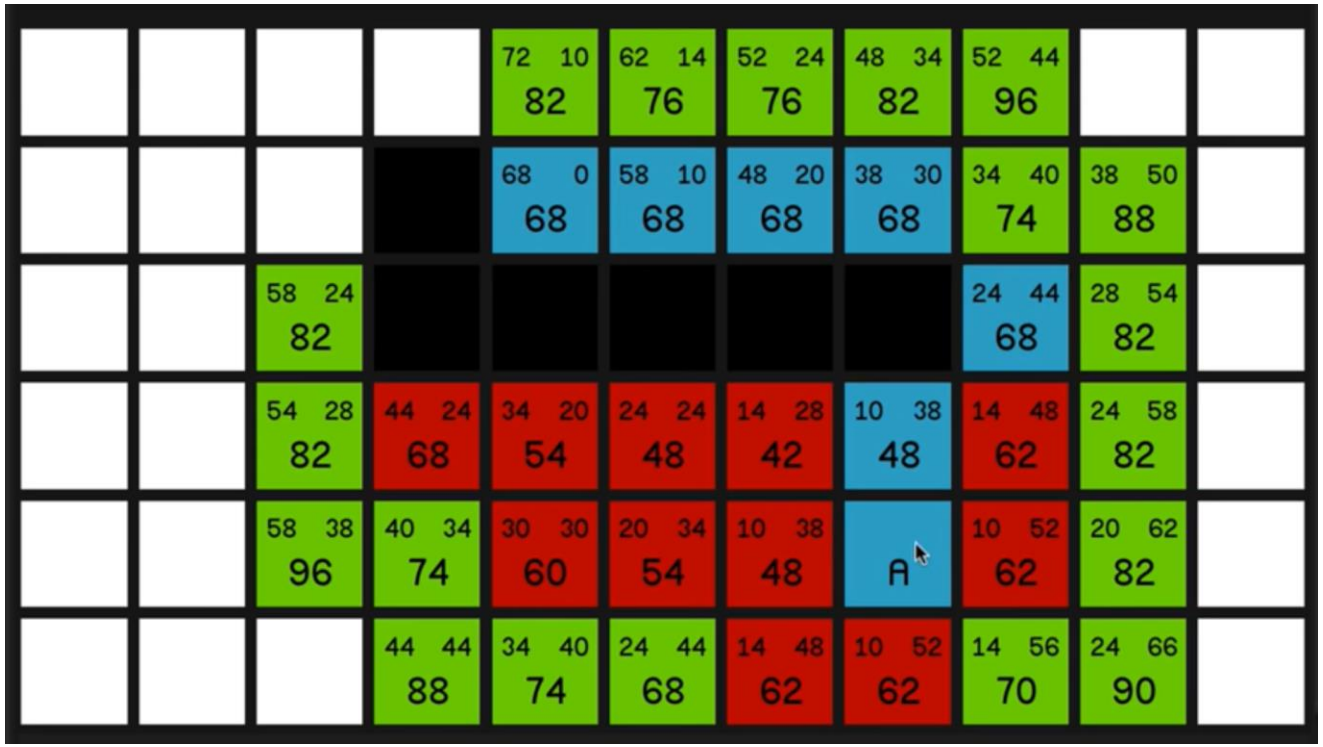
Algoritmo de path-finding e travessia de grafo.

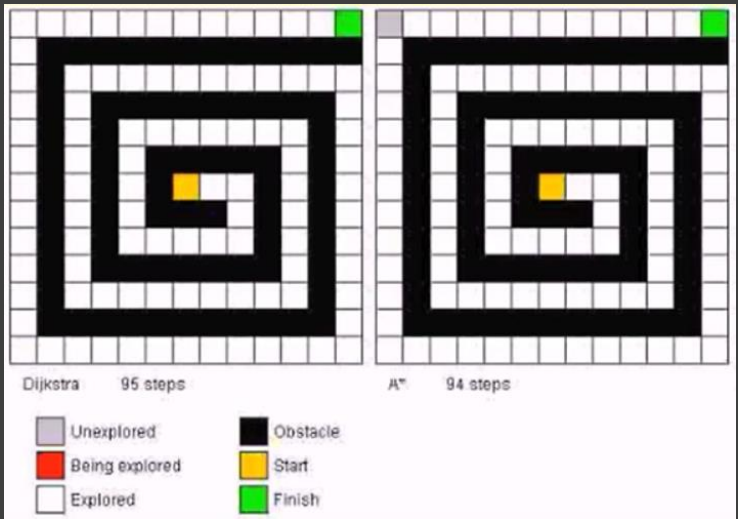
Bem eficiente.

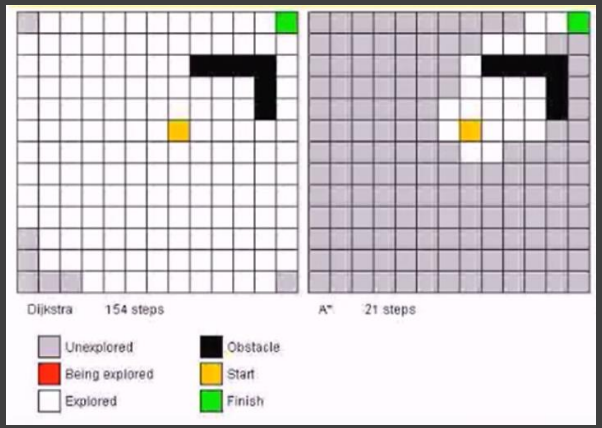
Bem inteligente.

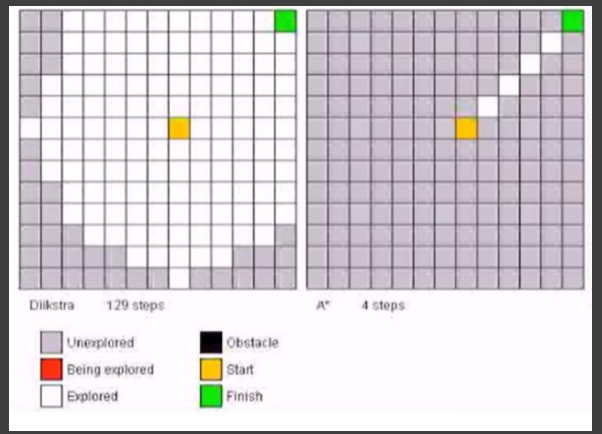
Funcionamento

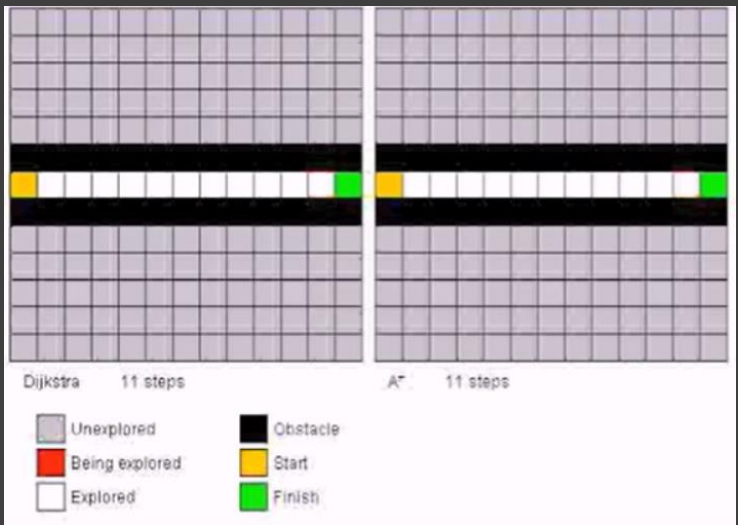
- Ponto A e Ponto B
- Vermelhos representam espaços utilizados
- Verdes representam a as possíveis opções
- A algoritmo passa nas menores somas
- Diagonais valem 14
- Verticais e horizontais 10




Dijkstra 95 steps
A* 94 steps
Unexplored
Being explored
Explored
Obstacle
Start
Finish


Dijkstra 154 steps
A* 21 steps
Unexplored
Being explored
Explored
Obstacle
Start
Finish


Dijkstra 129 steps
A* 4 steps
Unexplored
Being explored
Explored
Obstacle
Start
Finish


Dijkstra 11 steps
A* 11 steps
Unexplored
Being explored
Explored
Obstacle
Start
Finish

Desempenho

Geração do mapa

Cria matriz

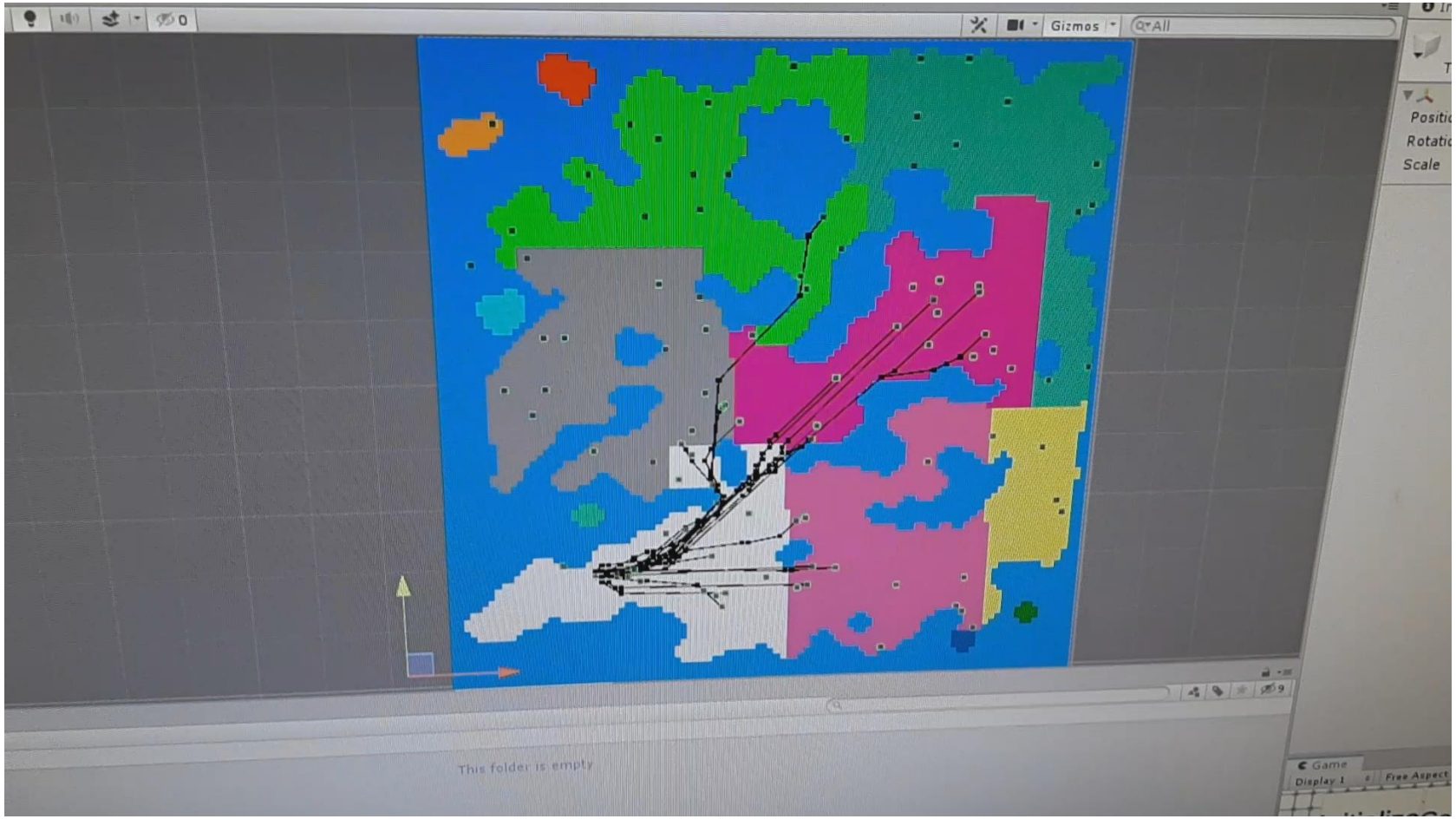
Faz uma separação e aglomeração de pontos

Adiciona Regiões ao mapa

Separa pontos pretos

Cria o target

Geração do mapa



Pseudo-código

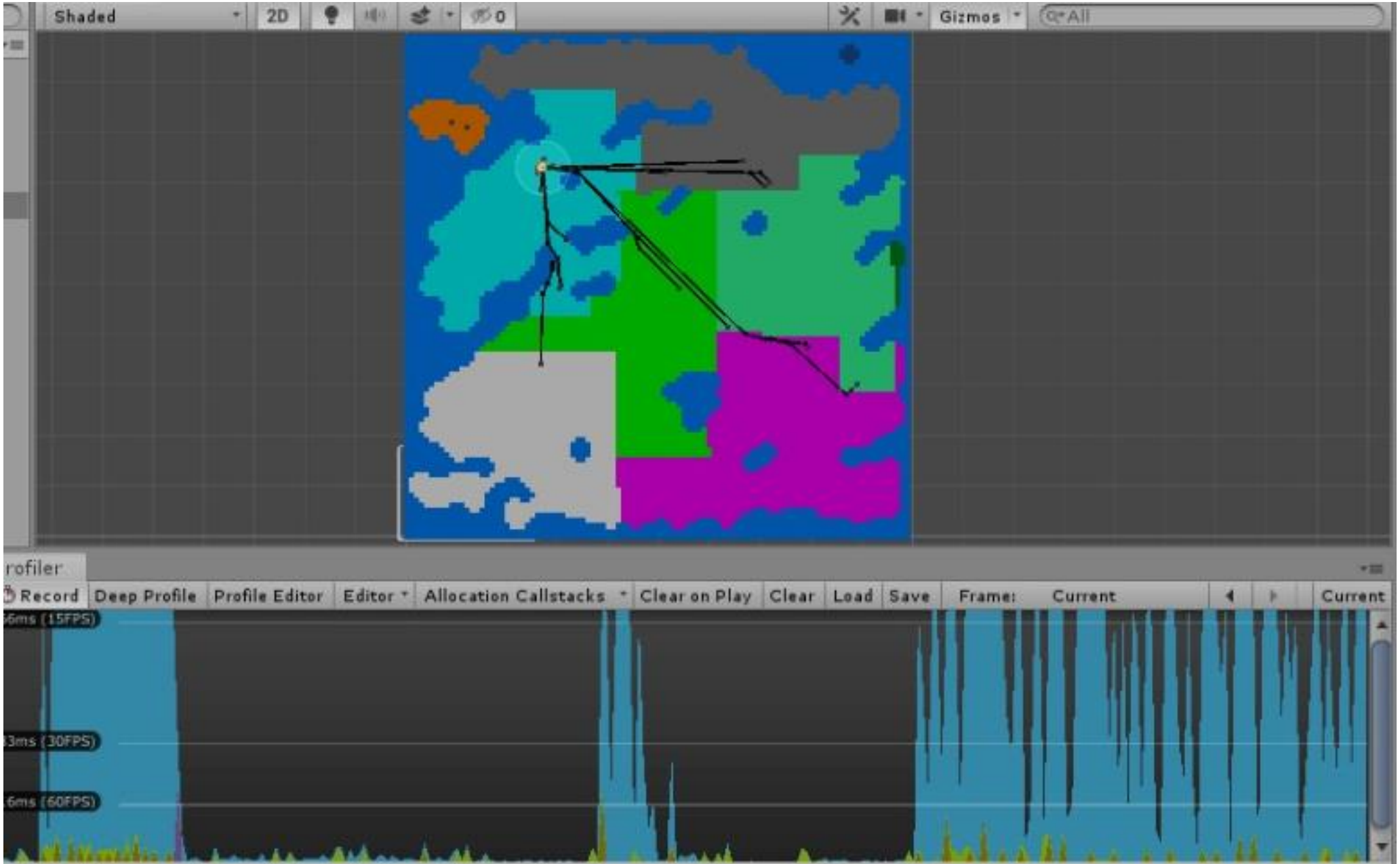
```
function reconstruct_path(cameFrom, current)
    total_path := {current}
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.prepend(current)
    return total_path



function A_Star(start, goal, h)


    openSet := {start}


    cameFrom := an empty map


    gScore := map with default value of Infinity
    gScore[start] := 0


    fScore := map with default value of Infinity
    fScore[start] := h(start)
```

```
while openSet is not empty
    current := the node in openSet having the lowest fScore[] value
    if current = goal
        return reconstruct_path(cameFrom, current)

    openSet.Remove(current)
    for each neighbor of current


        tentative_gScore := gScore[current] + d(current, neighbor)
        if tentative_gScore < gScore[neighbor]

            cameFrom[neighbor] := current
            gScore[neighbor] := tentative_gScore
            fScore[neighbor] := gScore[neighbor] + h(neighbor)
            if neighbor not in openSet
                openSet.add(neighbor)


return failure
```

Desempenho single-thread em lista

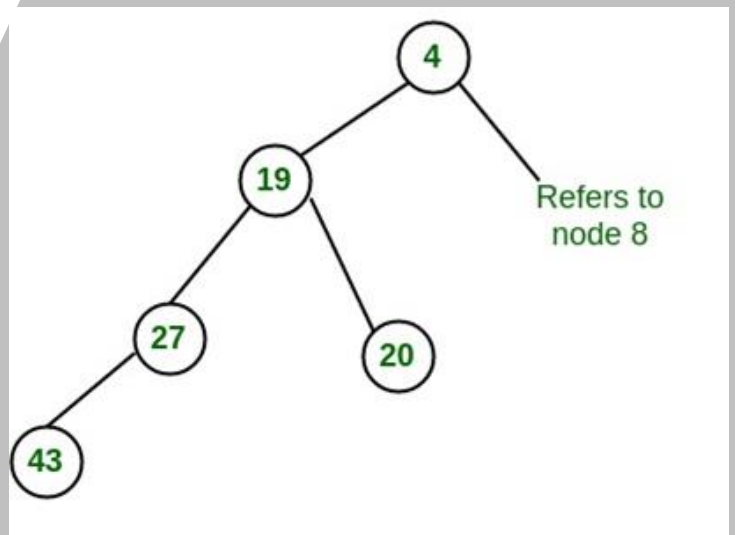
- Para o primeiro teste utilizamos uma lista para organizar os pontos ainda não visitados pelo caminho
- O programa prioriza os menores valores,esses valores são calculados levando em consideração a ditancia do destino e o "custo de deslocamento" , para assim achar o menor caminho até o ponto final.
- Esse método teve o pior desempenho devido à demora na troca de informações entre a lista e o programa.

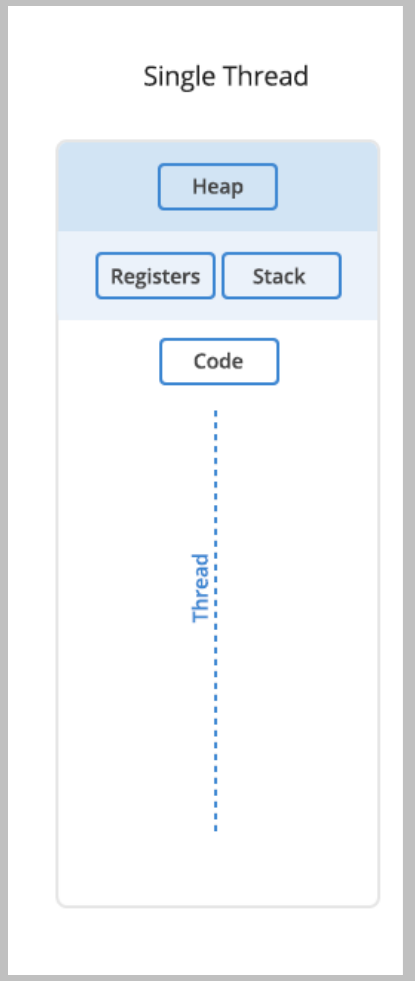
Contagem de FPS do programa



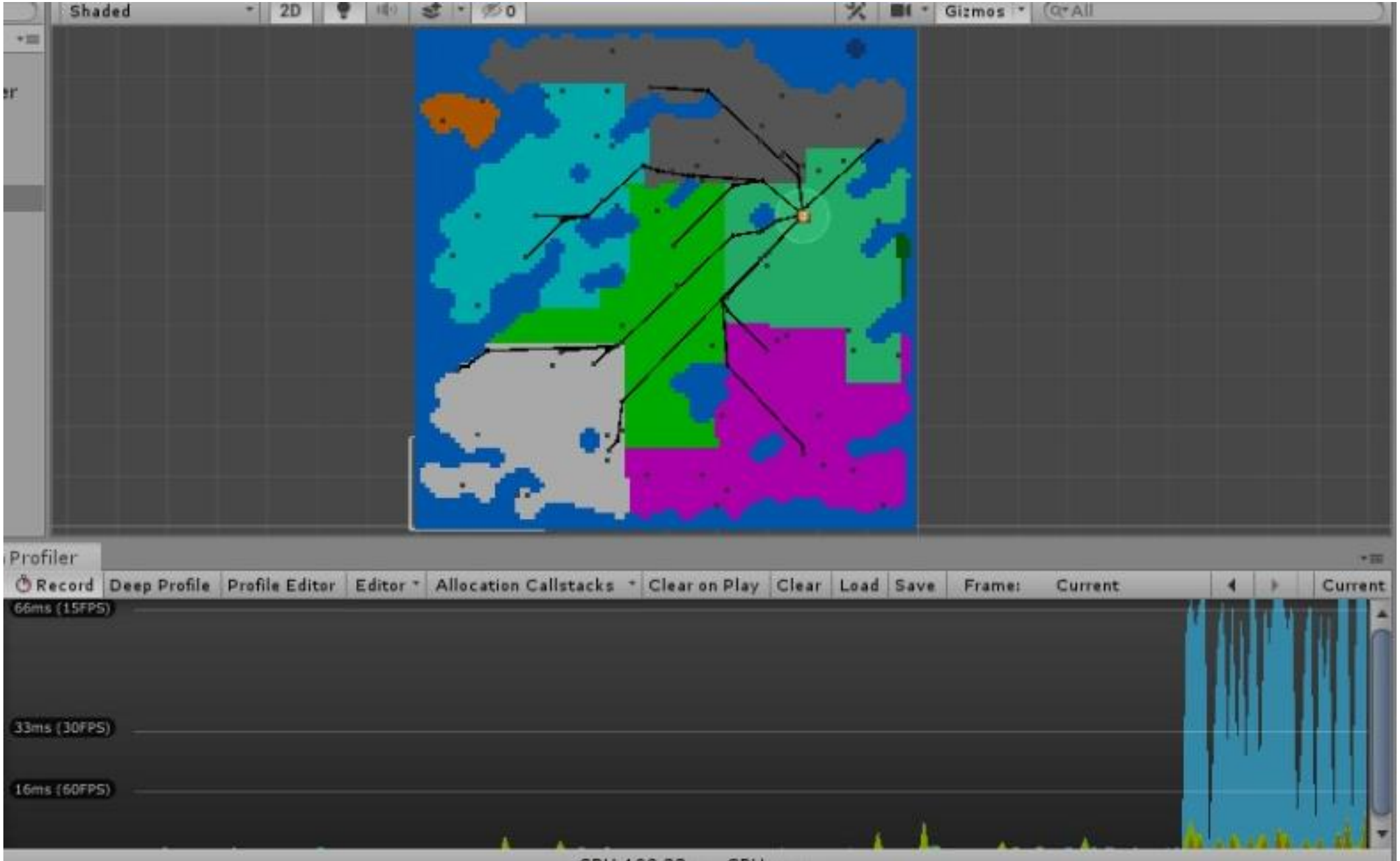
Desempenho single-thread otimizado

- Desta vez o programa utiliza conceitos de heap para organizar os pontos ainda não visitados.
- Ainda existe um tempo de espera, já que somente uma tread será responsável por todas as requisições, porem requer menor tempo já que o procedimento com heap permite obter o menor valor de forma mais rápida
- Algoritmo com desempenho razoável para bom.



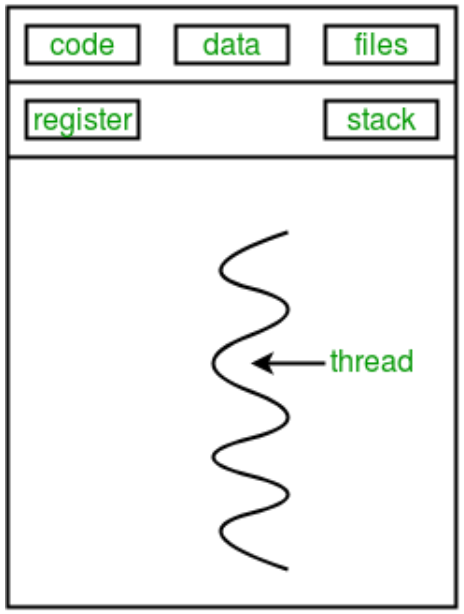


Contagem de FPS do programa

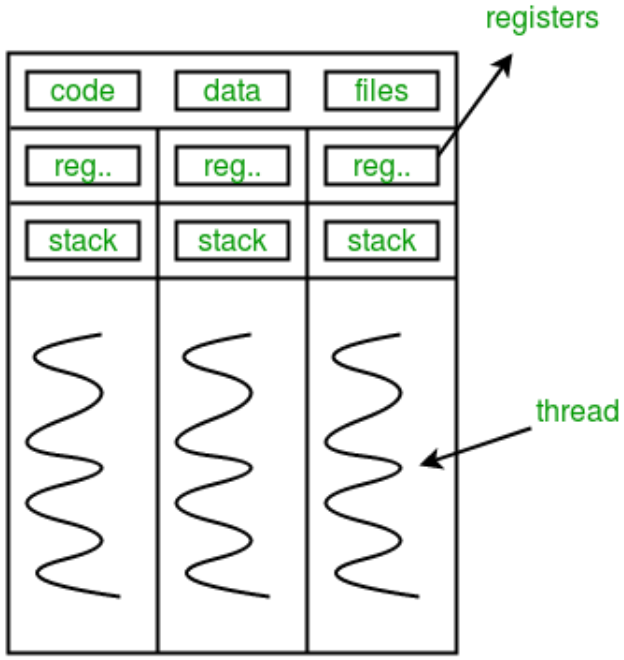


Melhoria Proposta

- A mesma versão otimizada do algoritmo será utilizada, utilizando uma estrutura Heap para armazenar o openSet e uma hash para o closedSet.
- Paralelização do processo, utilizando todas as requisições em varios threads simultâneamente.
- Algoritmo com desempenho bom para ótimo.
- Essa implementação também permite que outro processamento simultâneo do programa.

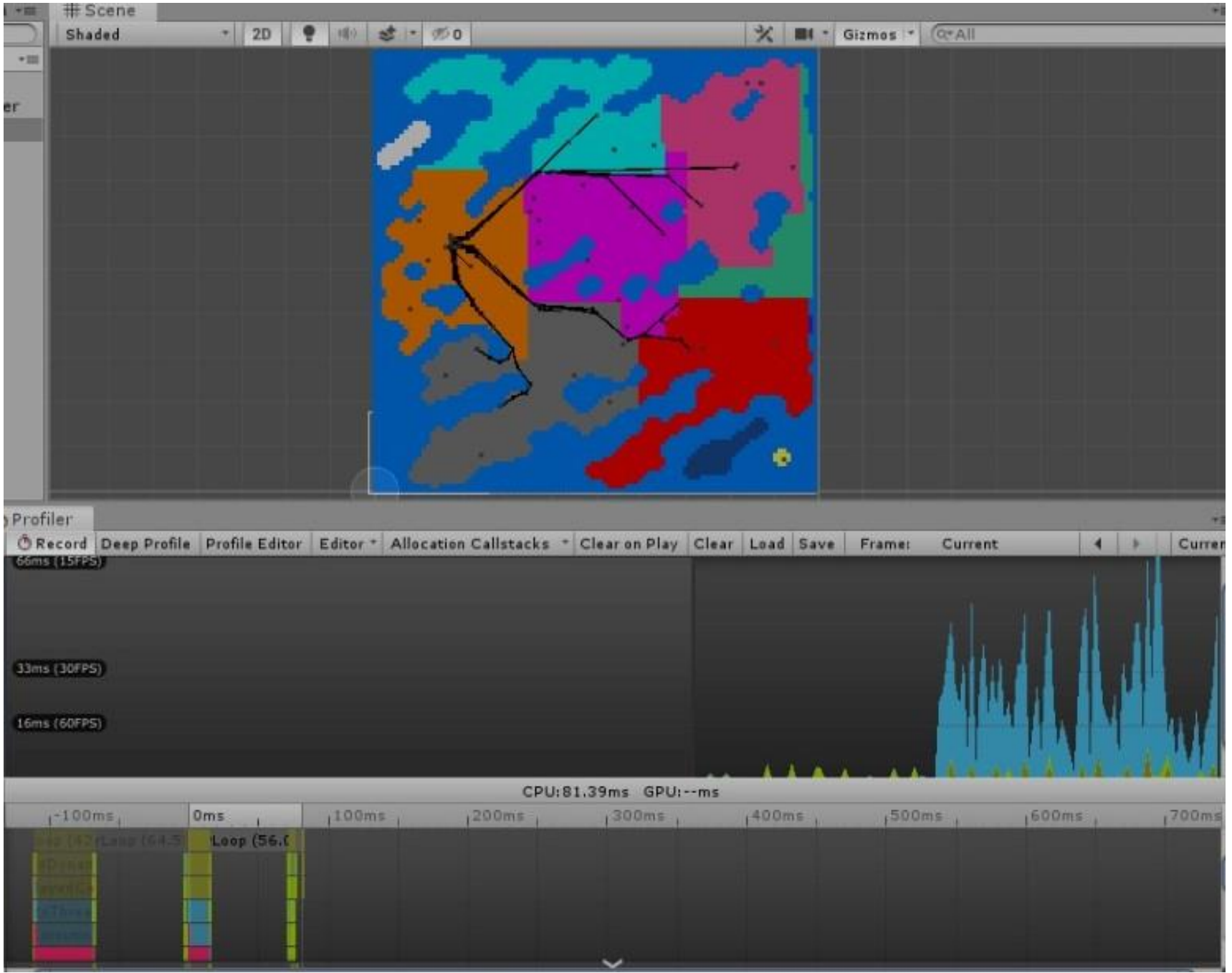


single-threaded process



multithreaded process

Contagem de FPS do programa



Bibliografia

- https://en.wikipedia.org/wiki/A*_search_algorithm 05/11/2019 23:15 A* Searching alghorithm, wikipedia
- https://www.youtube.com/watch?v=-L-WgKMFuhE 05/11/2019 18:23, A* Pathfinding (E01: algorithm explanation), youtube
- http://www.ic.unicamp.br/~norton/arquivos/alg_grafos.pdf , Complexidade de Algoritmos I – Algoritmos em Grafos