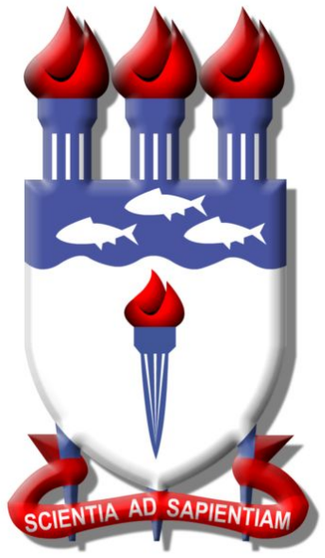


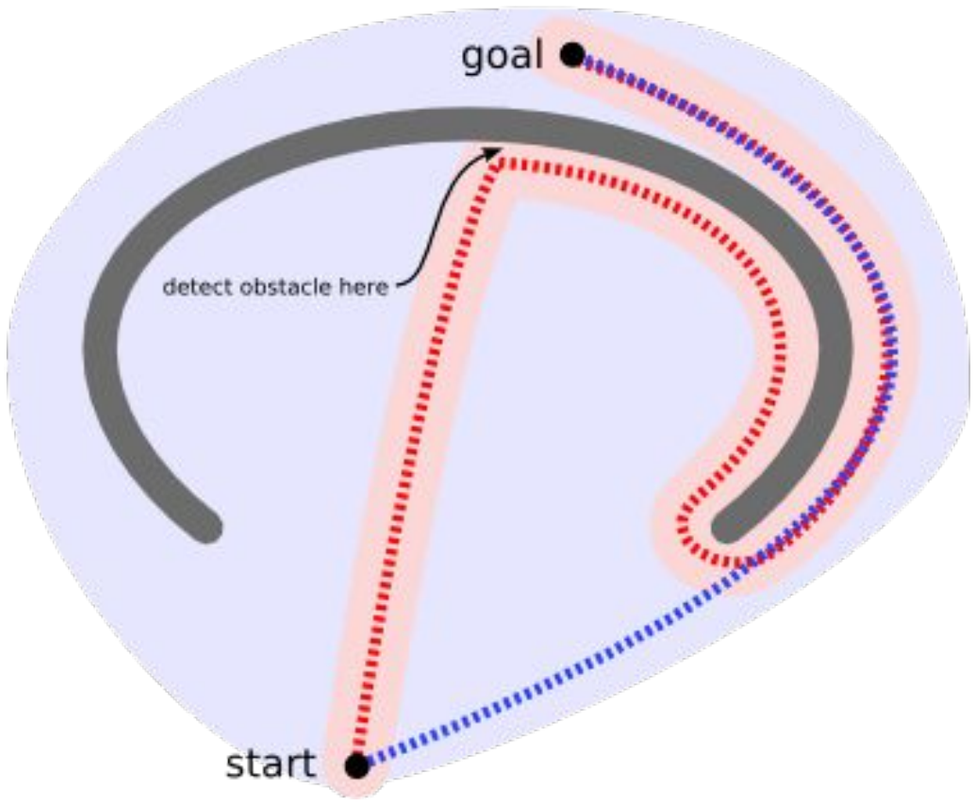
Algoritmo A*

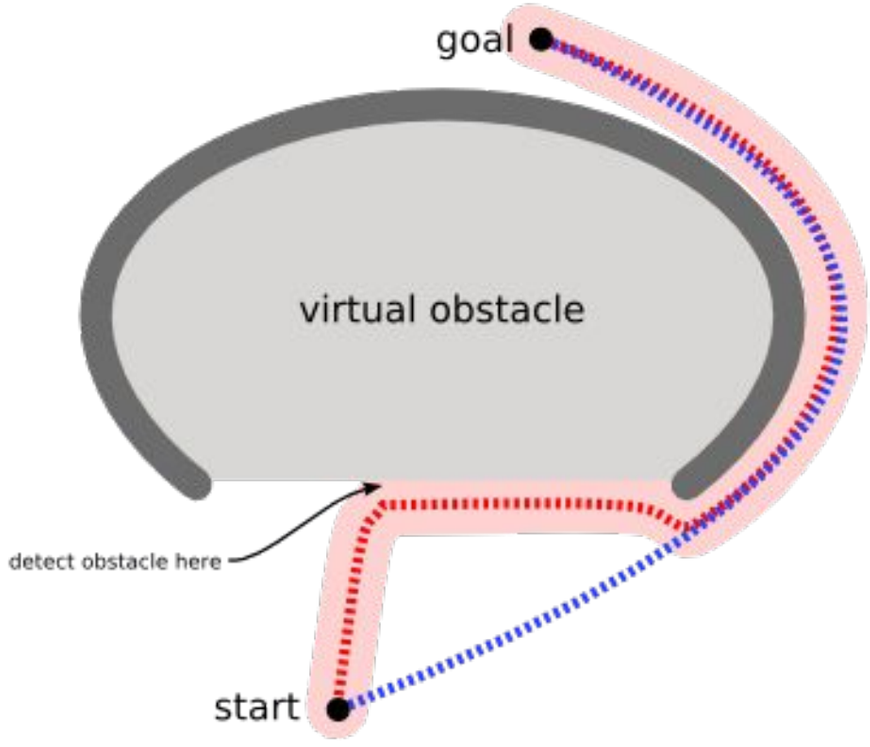
Equipe: Jackson Bruno,Lucas Vinícius,Vinícius Rafael e Victor Araújo.

Link do github do projeto
https://github.com/nivuciis/PROJETO_ESTRUTURA_DE_DADOS

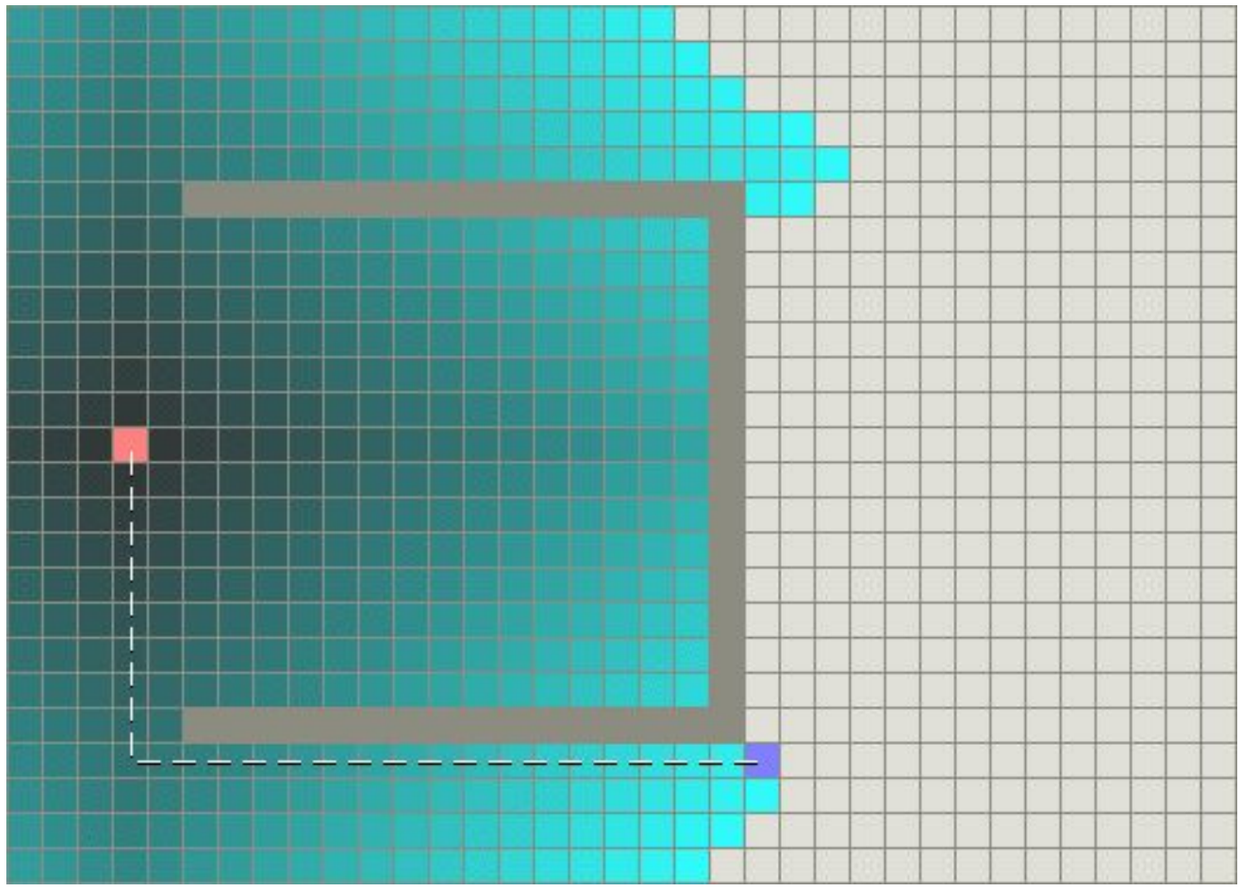
Problema

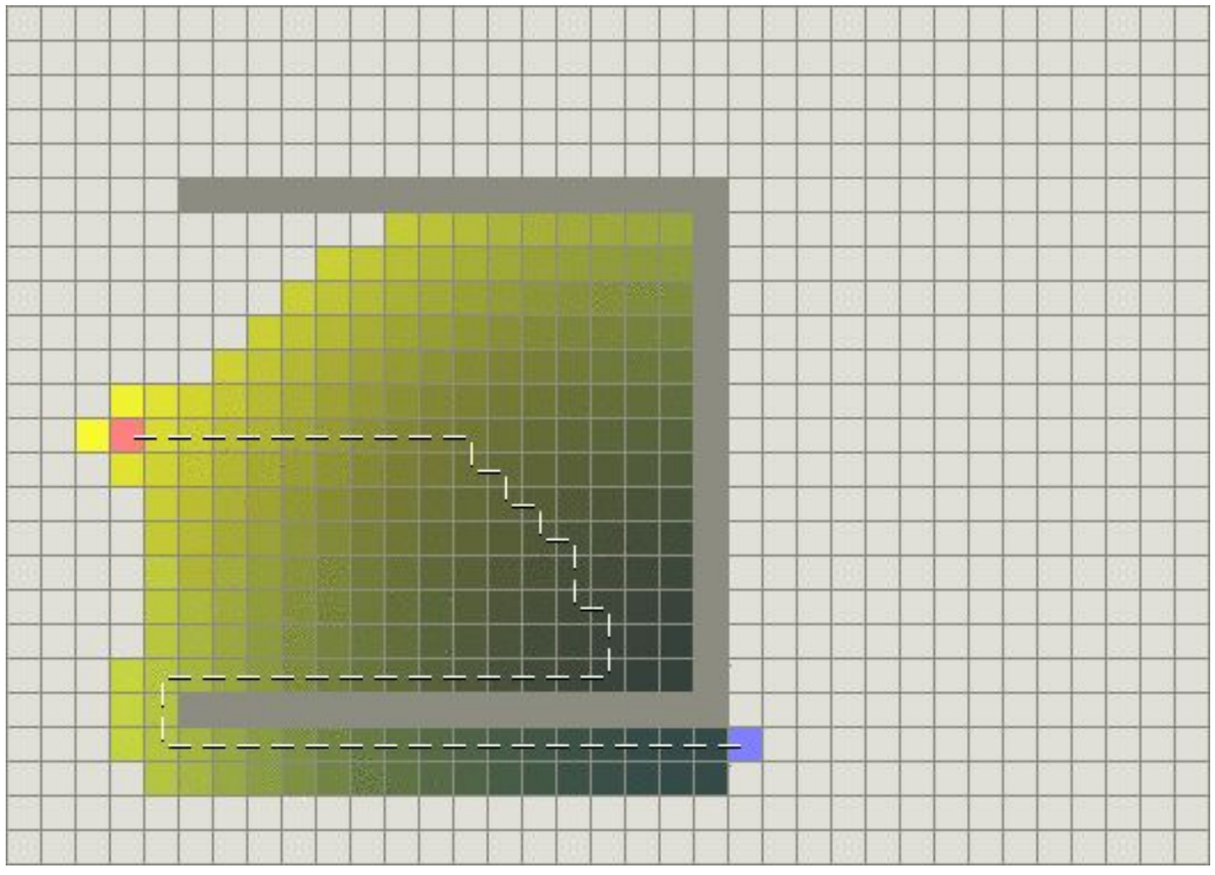
- **O movimento para um único objeto parece fácil. Pathfinding é complexo. Por que se preocupar com pathfinding?**
- **Considere a seguinte situação:**





Algoritmo de Dijkstra e Best-First-Search

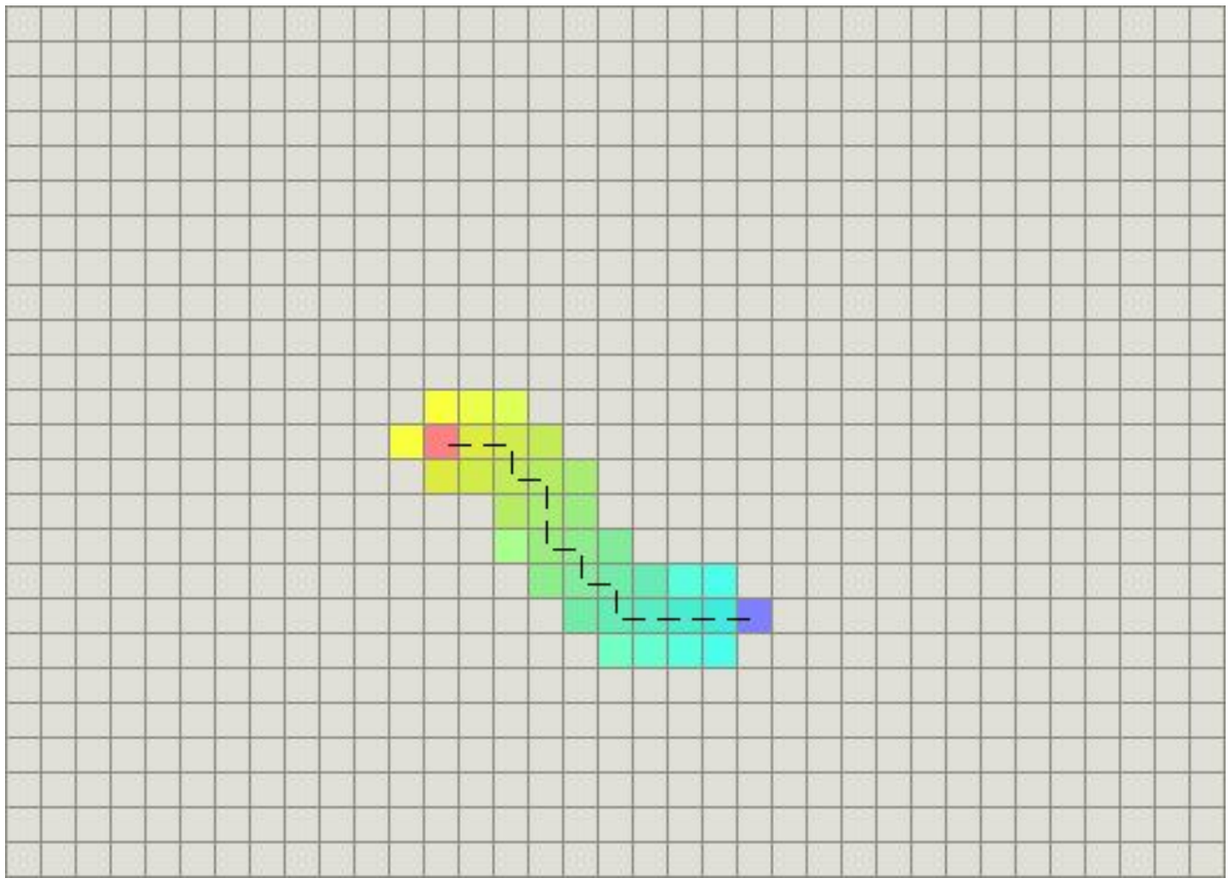


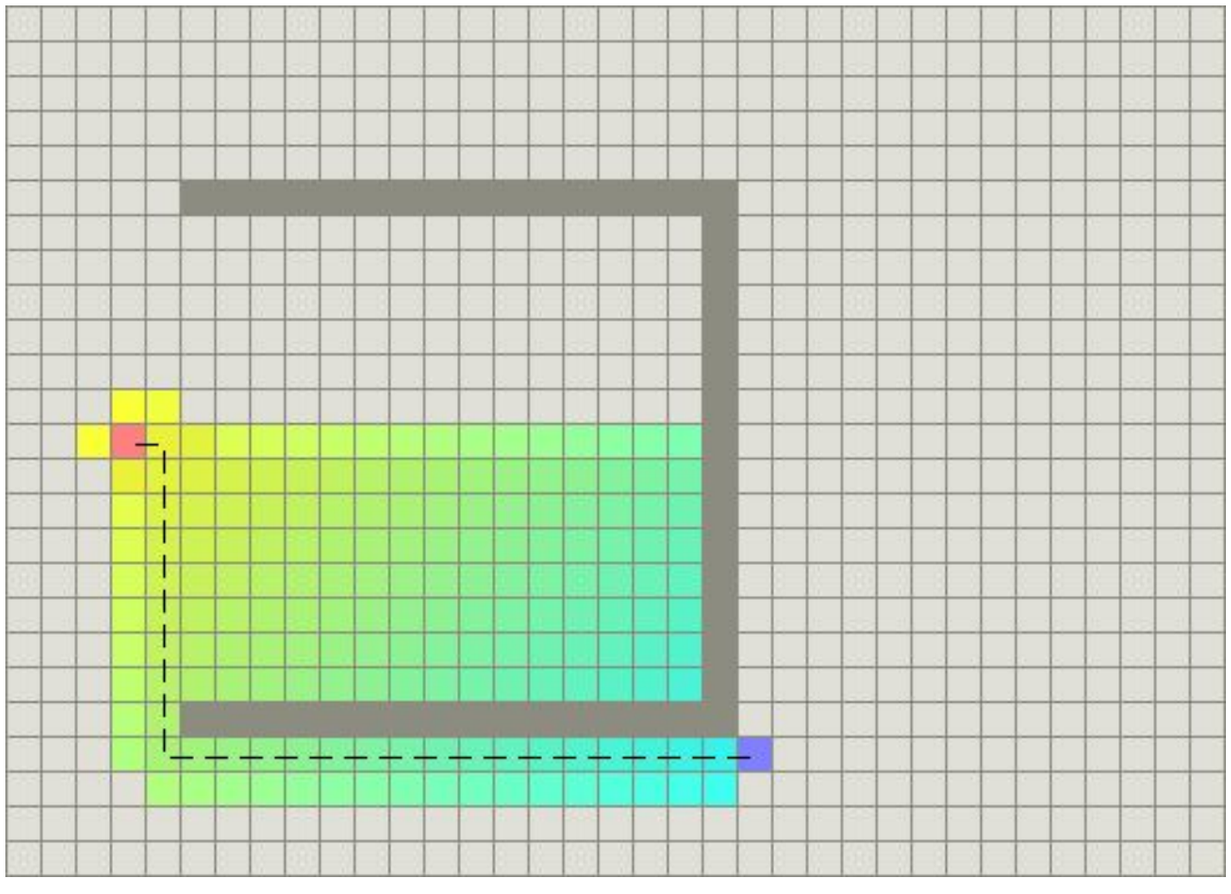


Não seria bom combinar a melhor das informações?

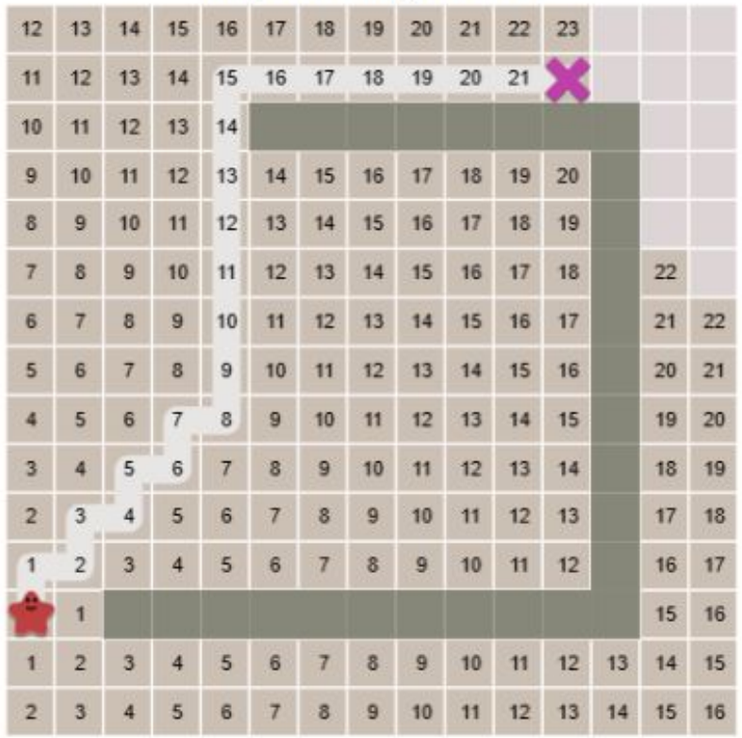
Algoritmo A*

- **A* é como o Algoritmo de Dijkstra, pois pode ser usado para encontrar um caminho mais curto.**
- **A* é como Best-First-Search, pois pode usar uma heurística para se guiar.**

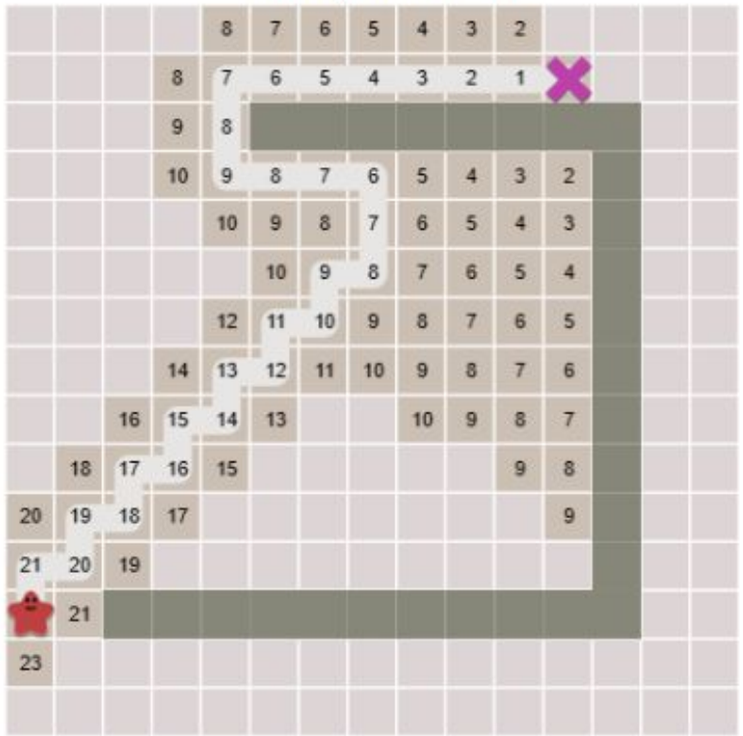




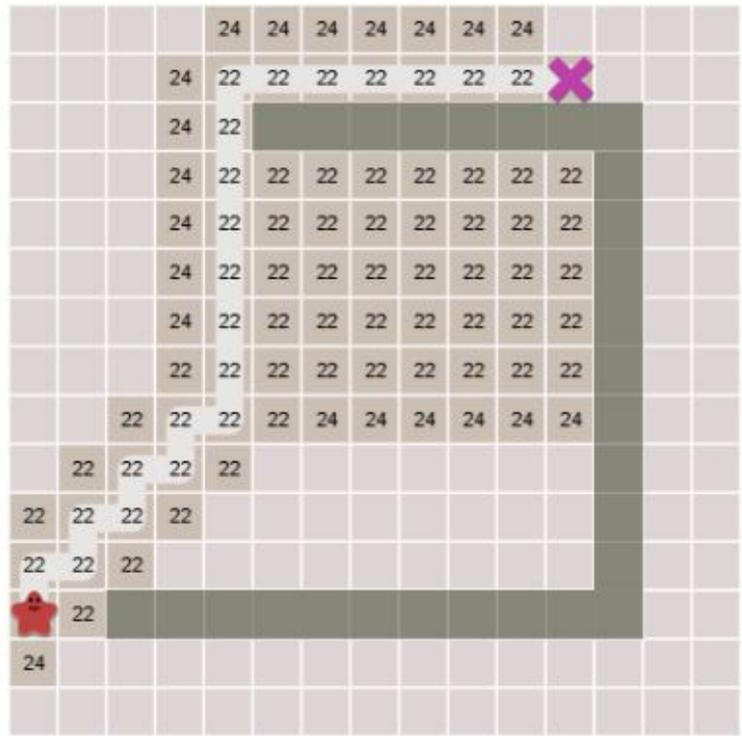
Dijkstra's Algorithm



Greedy Best-First



A* Search



A* ADT

```
struct ASPath {...}
struct Node {...}
struct VisitedNodes {...}
void SetNodeIsGoal(Node n);
SetNodeEstimatedCost (Node n,float EstimatedCost);
```

Estruturas / Funções

```
struct __ASPath {
    size_t nodeSize;
    size_t count;
    float cost;
    int8_t nodeKeys[];
};

typedef struct {
    unsigned isClosed:1;
    unsigned isGoal:1;
    unsigned hasEstimatedCost:1;
    float estimatedCost;
    float cost;
    size_t openIndex;
    size_t parentIndex;
    int8_t nodeKey[];
} NodeRecord;

struct VisitedNodes {
    const ASPathNodeSource *source;
    void *nodeRecords;
};

struct Node{
    VisitedNodes nodes;
    size_t index;
}
```

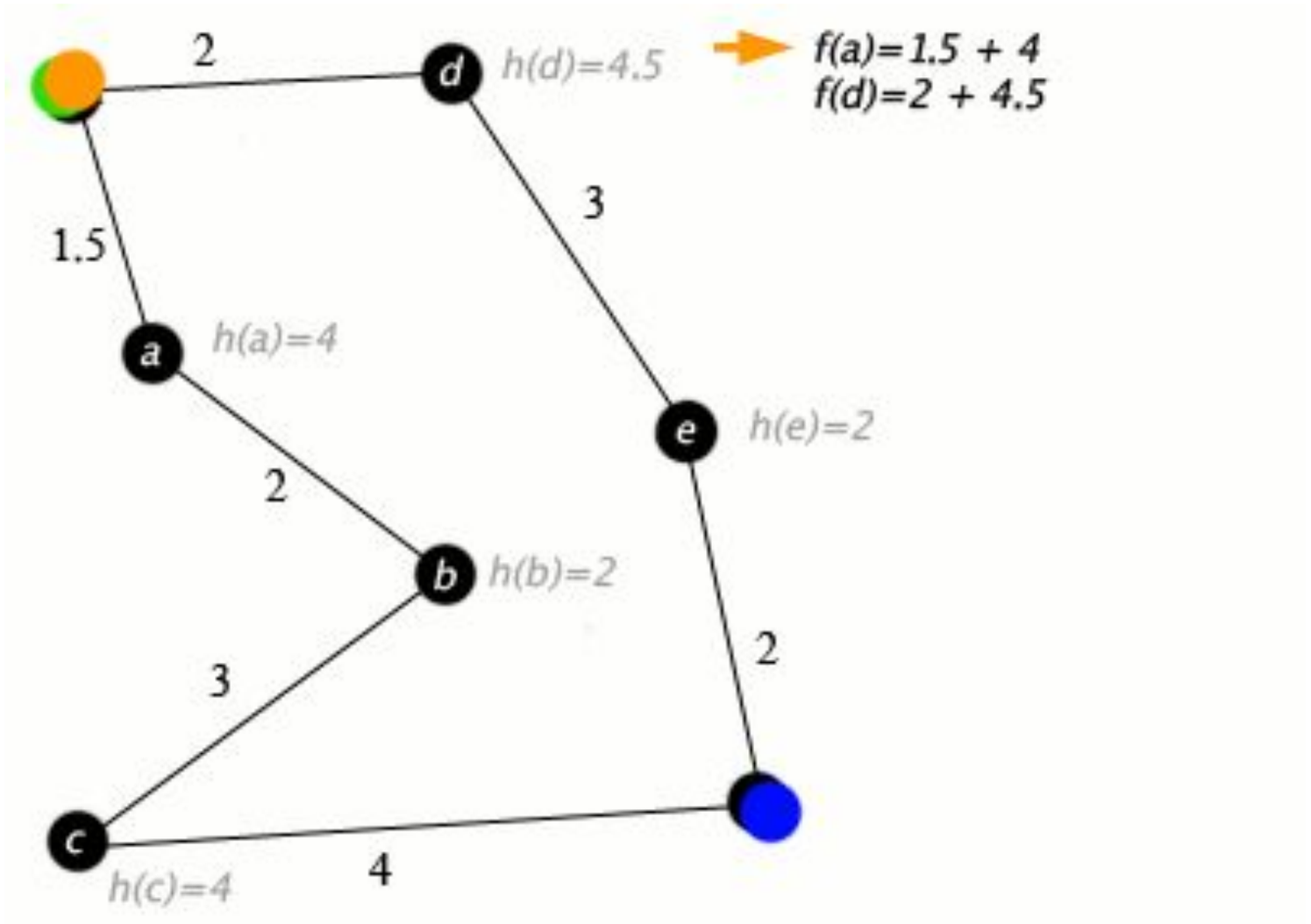
```
void SetNodeIsGoal(Node n)
{
    if (!NodeIsNull(n)) {
        NodeGetRecord(n)->isGoal = 1;
    }
}
```

```
void SetNodeEstimatedCost(Node n, float estimatedCost)
{
    NodeRecord *record = NodeGetRecord(n);
    record->estimatedCost = estimatedCost;
    // estimatedCost é o custo estimado ( heurística ) dado por outra função
    record->hasEstimatedCost = 1;
}
```

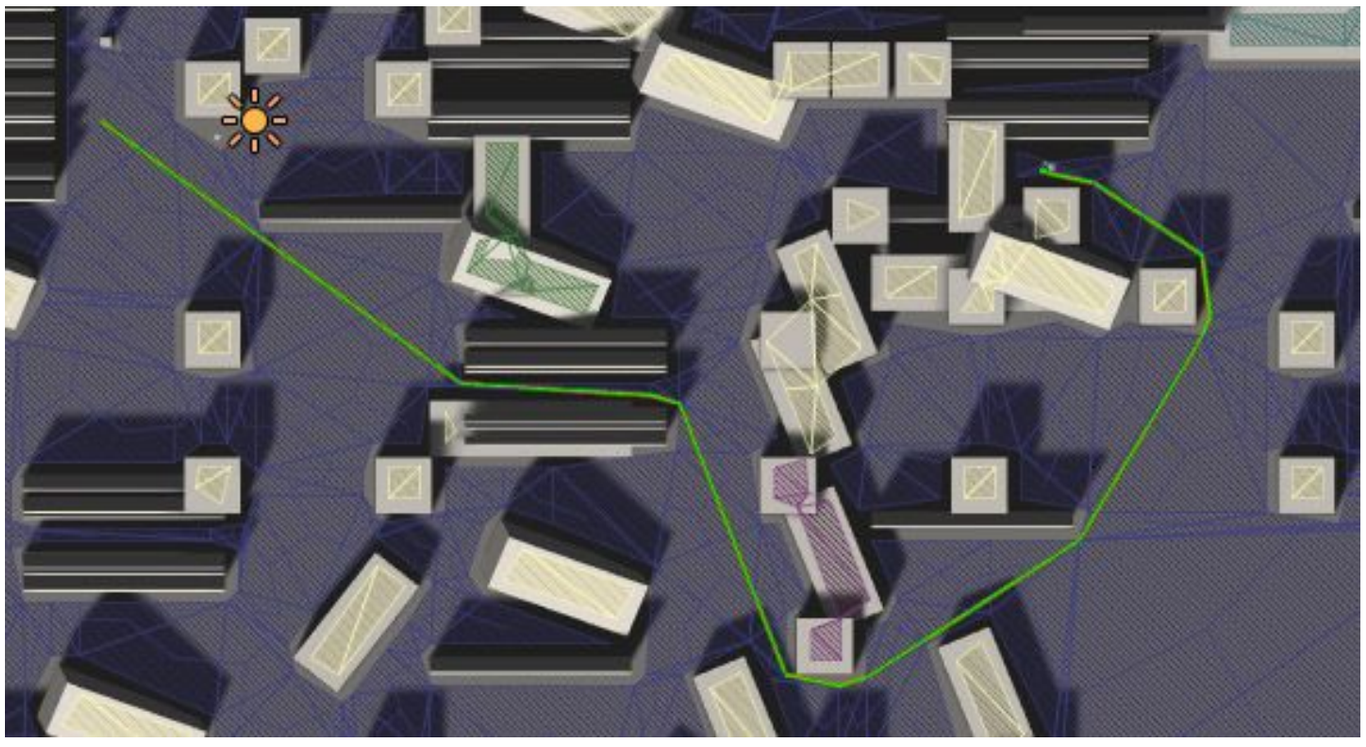
Código-base e cerne do A*

```
ASPath ASPathCreate(const ASPathNodeSource *source, void *context, void *startNodeKey, void *goalNodeKey)
{
    if (!startNodeKey || !source || !source->nodeNeighbors || source->nodeSize == 0) {
        return NULL;
    }

    Node current = GetNode(visitedNodes, startNodeKey);
    Node goalNode = GetNode(visitedNodes, goalNodeKey);
    int count = 0;
    ASPath path = NULL;

    SetNodeIsGoal(goalNode);
    SetNodeEstimatedCost(current,  GetPathCostHeuristic(current, goalNode));

    while ( !NodeIsGoal(current) ) {

        count++;
        const int shouldExit = source->earlyExit(visitedNodes->nodeRecordsCount, GetNodeKey(current), goalNodeKey, context);

        if (shouldExit)
        {
            SetNodeIsGoal(current);
            break;
        }

        current = GetOpenNode(visitedNodes);
    }

    path = malloc(sizeof(struct __ASPath) + (count * source->nodeSize));
    path->nodeSize = source->nodeSize;
    path->count = count;
    path->cost = GetNodeCost(current);
    return path;
}
```

Animação



Aplicações



Referências

Introduction to A* (link) -> (stanford.edu)

A* code -> AStar/AStar.c at master · BigZaphod/AStar · GitHub