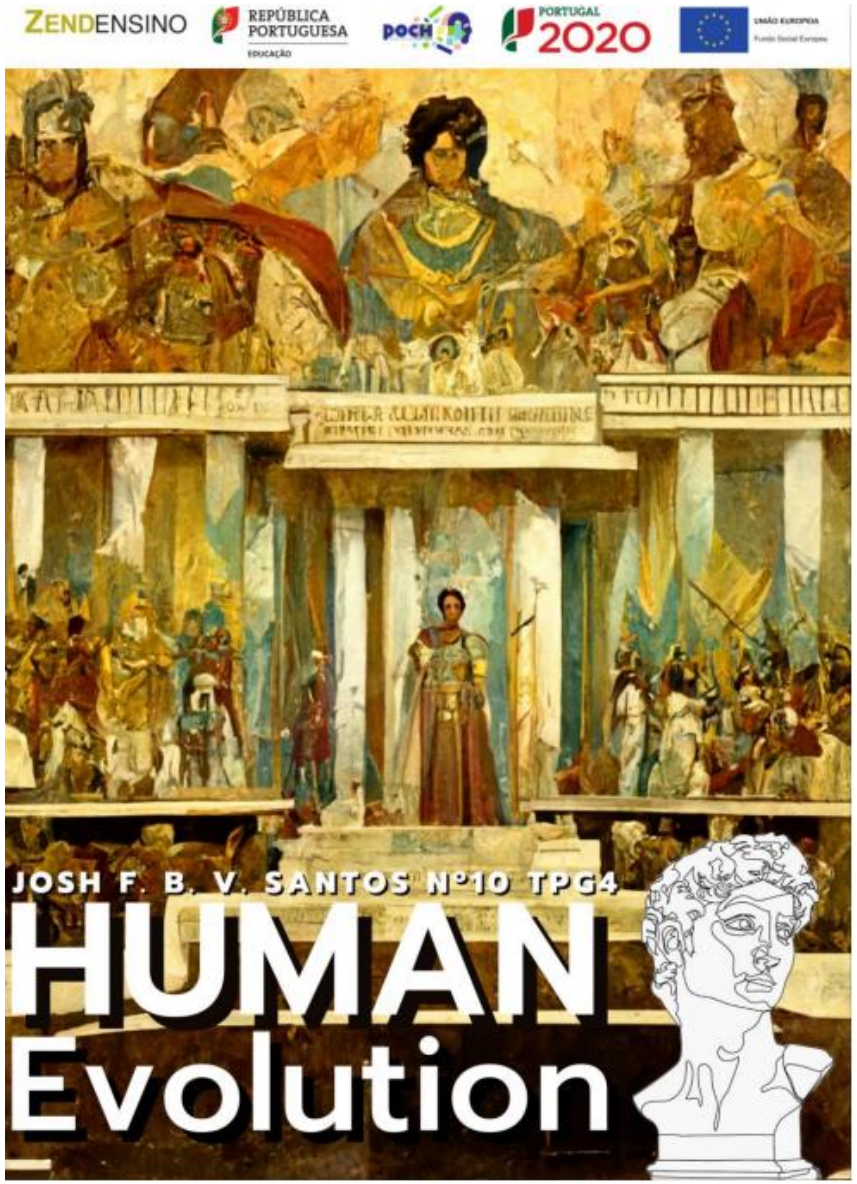


Human Evolution Jogo de Estratégia

Josh Santos Nº10 TGP4

Índice

Introdução:

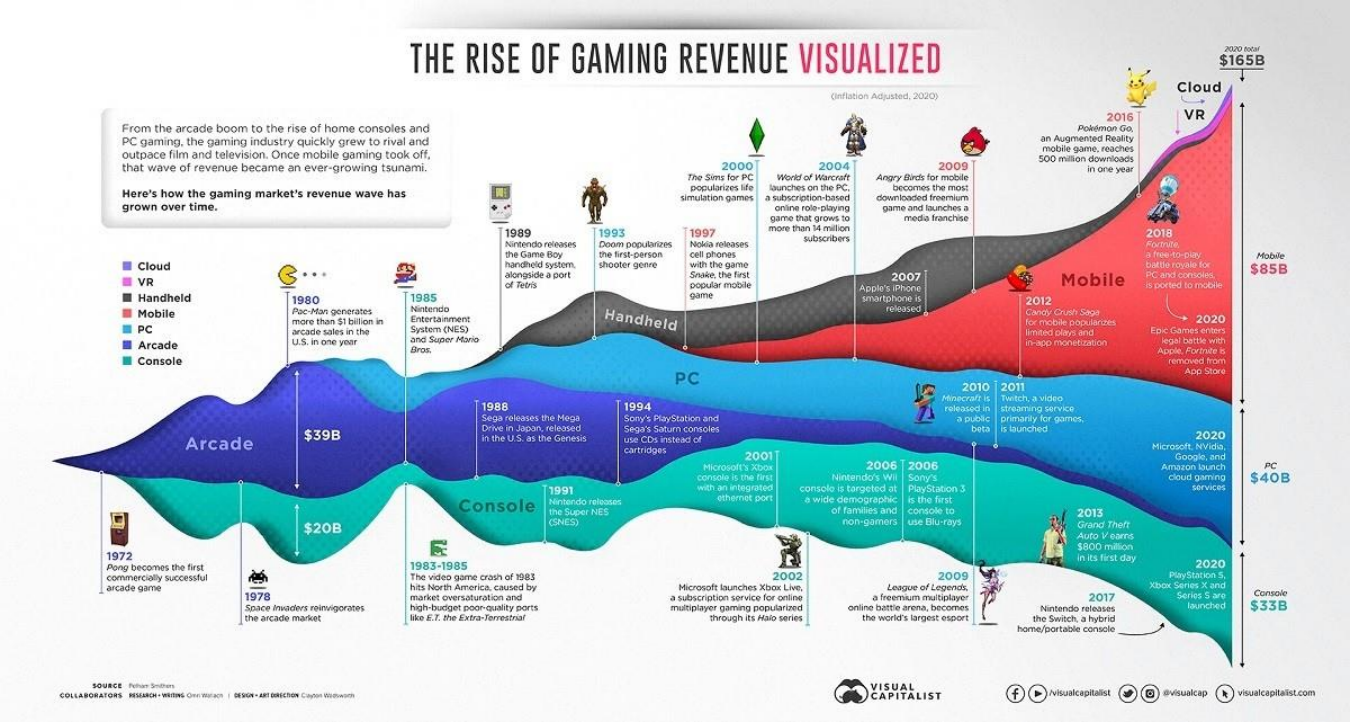
Porquê videojogos
História dos jogos
Género de videojogos 4x Civilization

Parte teórica:

O hexágono;
A grelha hexagonal;
Diferentes tipos de coordenadas numa grelha hexagonal;
Propriedades individuais de cada hexágono.
Criação do mapa.
Movimento de unidades.
Adjacência.
Heurística
Construções.

Porquê videojogos?

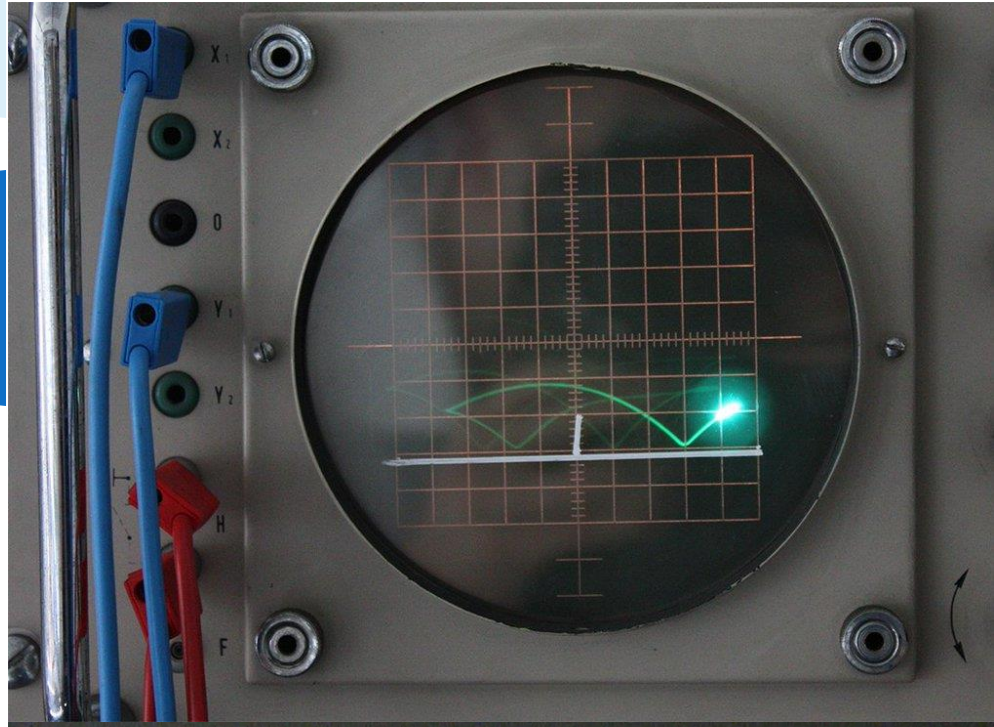
O tema dos videojogos interessa-me, tem subido em popularidade, sendo também uma das melhores formas de mostrar aptidão em diferentes áreas da informática



História dos videojogos

Os videojogos começaram a aparecer nos 1950's

e 1960's com o *CRT Amusement device* e *Spacewar!* Posteriormente os jogos começaram a ficar mais complicados, passando para *arcade,* consolas domésticas, consolas portáteis, PC's e telemóveis.



Género de videojogos 4x

Na indústria dos videojogos existem vários géneros, decidi fazer um jogo 4x, uma subcategoria dos jogos de estratégia em que lideras um império, 4x significando *eXplorar, eXpandir, eXtrair, eXterminar.*

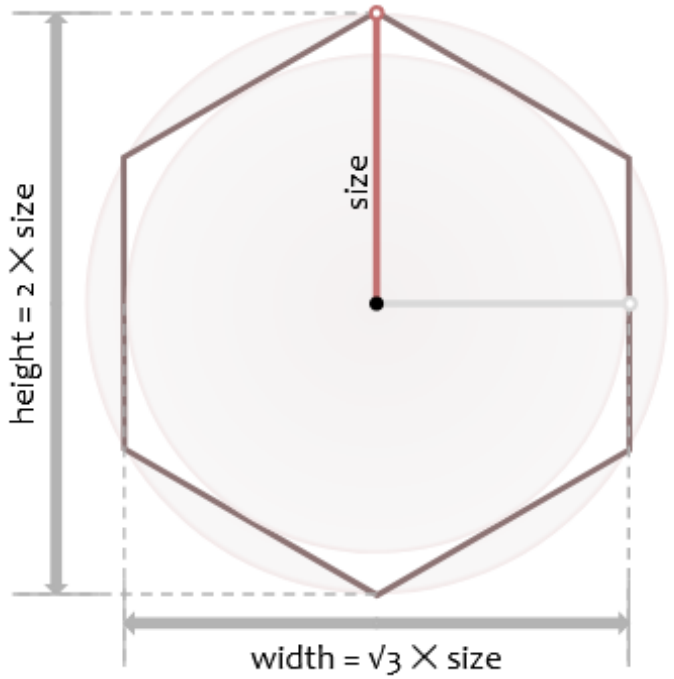
Civilization

Civilization é uma das franquias mais influentes na categoria 4x, daí inspirar-me na instalação mais recente da mesma, Civilization VI.

Parte teórica

A parte teórica divide-se nas seguintes partes:

- O hexágono;
- A grelha hexagonal;
- Diferentes tipos de coordenadas numa grelha hexagonal;
- Propriedades individuais de cada hexágono.
- Criação do mapa.
- Movimento das unidades.
- Adjacência.
- Heurística
- Construções e Unidades.
- Cidades-Estado




```
Script de Unity | 6 referências
public class HexMetrics : MonoBehaviour
{
    public const float outerRadius = 10f;

    public const float innerRadius = outerRadius * 0.866025404f; //half of sqrt3 approx.

    public static Vector3[] corners = {
        new Vector3(0f, 0f, outerRadius),
        new Vector3(innerRadius, 0f, 0.5f * outerRadius),
        new Vector3(innerRadius, 0f, -0.5f * outerRadius),
        new Vector3(0f, 0f, -outerRadius),
        new Vector3(-innerRadius, 0f, -0.5f * outerRadius),
        new Vector3(-innerRadius, 0f, 0.5f * outerRadius),
        new Vector3(0f, 0f, outerRadius)
    };
}
```

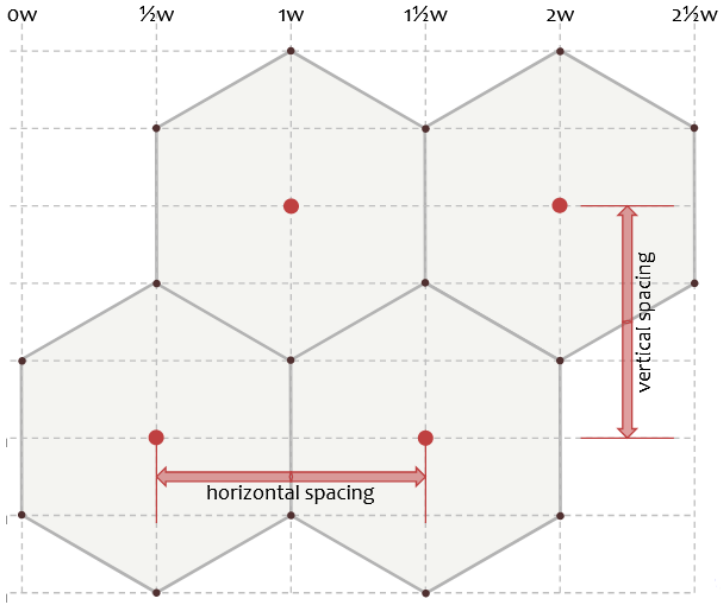
O hexágono

As medidas de um hexágono são defenidas por duas circunferências:

- A circunferência interior;
- A circunferência exterior.

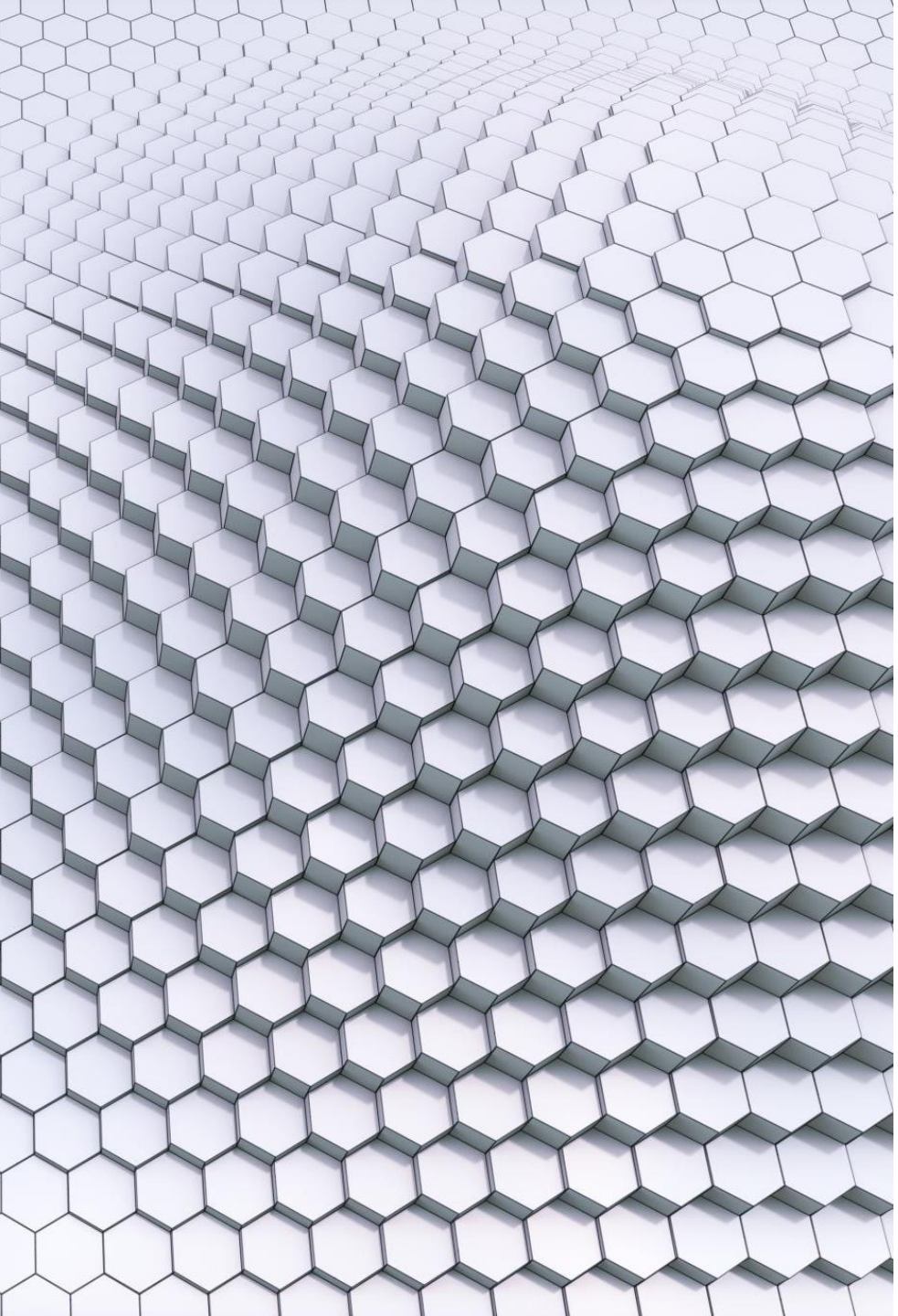
A grelha hexagonal

- Abaixo podemos ver um exemplo de uma grelha constituída por 4 hexágonos e o código usado para implementar a mesma.




```
cells = new HexCell[height * width];

for (int z = 0, i = 0; z < height; z++)
{
    for (int x = 0; x < width; x++)
    {
        CreateCell(x, z, i++);
    }
}
```

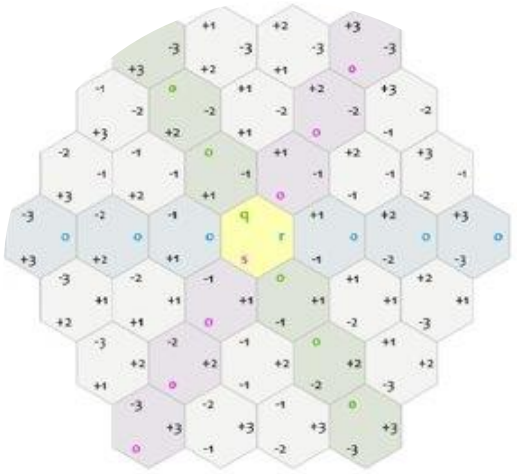
```
void CreateCell(int x, int z, int i)
{
    Vector3 position;
    position.x = (x + z * 0.5f - z / 2) * (HexMetrics.innerRadius * 2f);
    position.y = 0f;
    position.z = z * (HexMetrics.outerRadius * 1.5f);

    HexCell cell = cells[i] = Instantiate<HexCell>(cellPrefab);
    cell.transform.SetParent(transform, false);
    cell.transform.localPosition = position;
    cell.coordinates = HexCoordinates.FromOffsetCoordinates(x, z);
    cell.color = defaultColor;

    TextMeshProUGUI label = Instantiate<TextMeshProUGUI>(cellLabelPrefab);
    label.rectTransform.SetParent(gridCanvas.transform, false);
    label.rectTransform.anchoredPosition = new Vector2(position.x, position.z);
    label.text = cell.coordinates.ToStringOnSeparateLines();
}
```

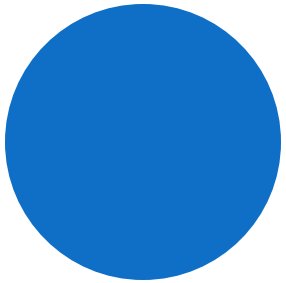


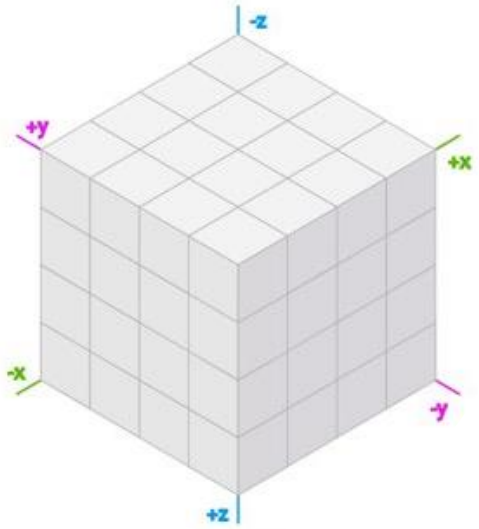
Coordenadas numa grelha hexagonal

Numa grelha hexagonal existem vários tipos de coordenadas que podem ser utilizadas, estas sendo:

- Coordenadas de cubo;
- Coordenadas axiais;
- Coordenadas offset;

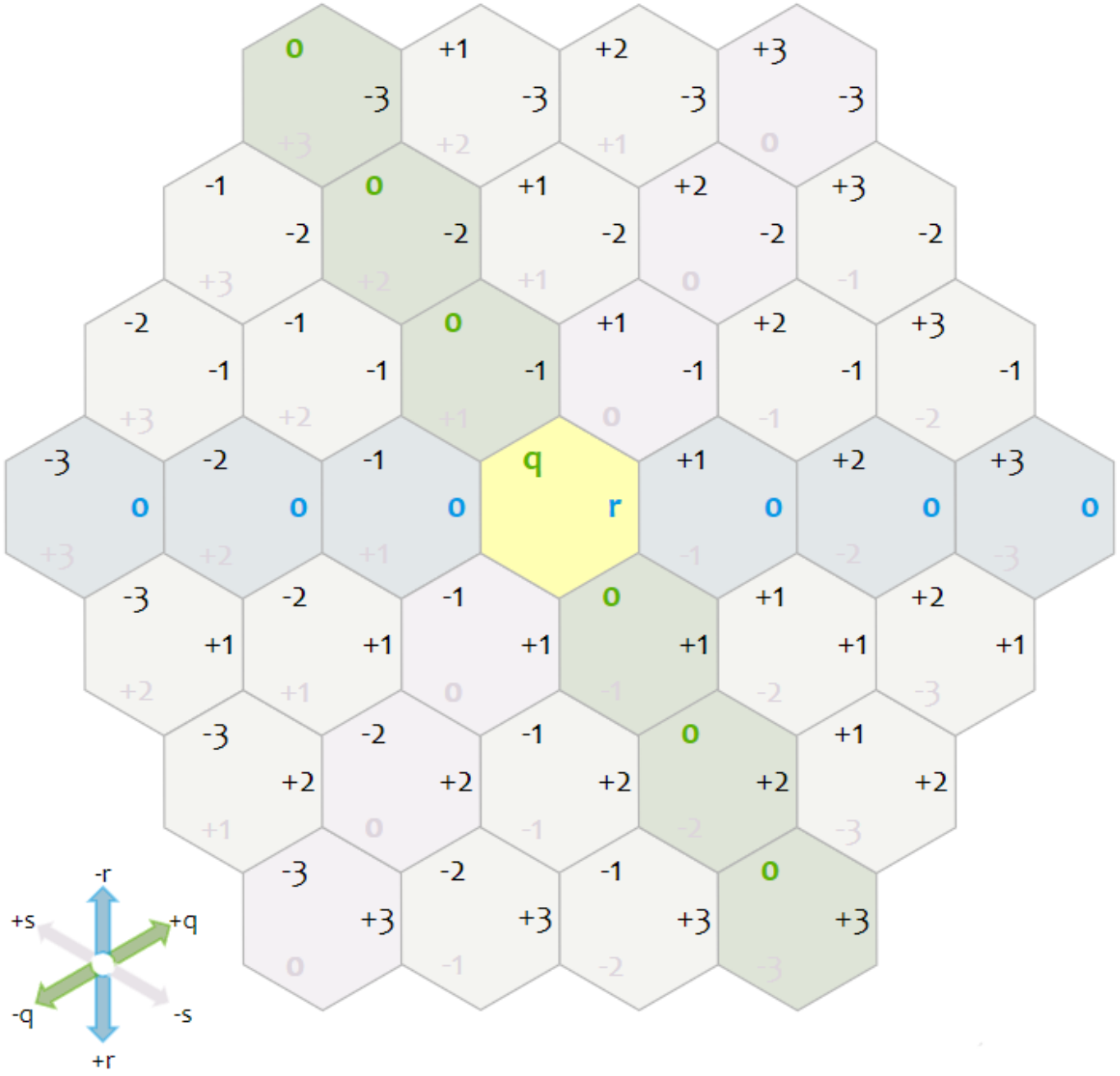






Coordenadas de cubo

- As coordenadas de cubo imaginam um hexágono como um cubo visto de um específico ângulo, utilizando então os vértices do mesmo para calcular as coordenadas

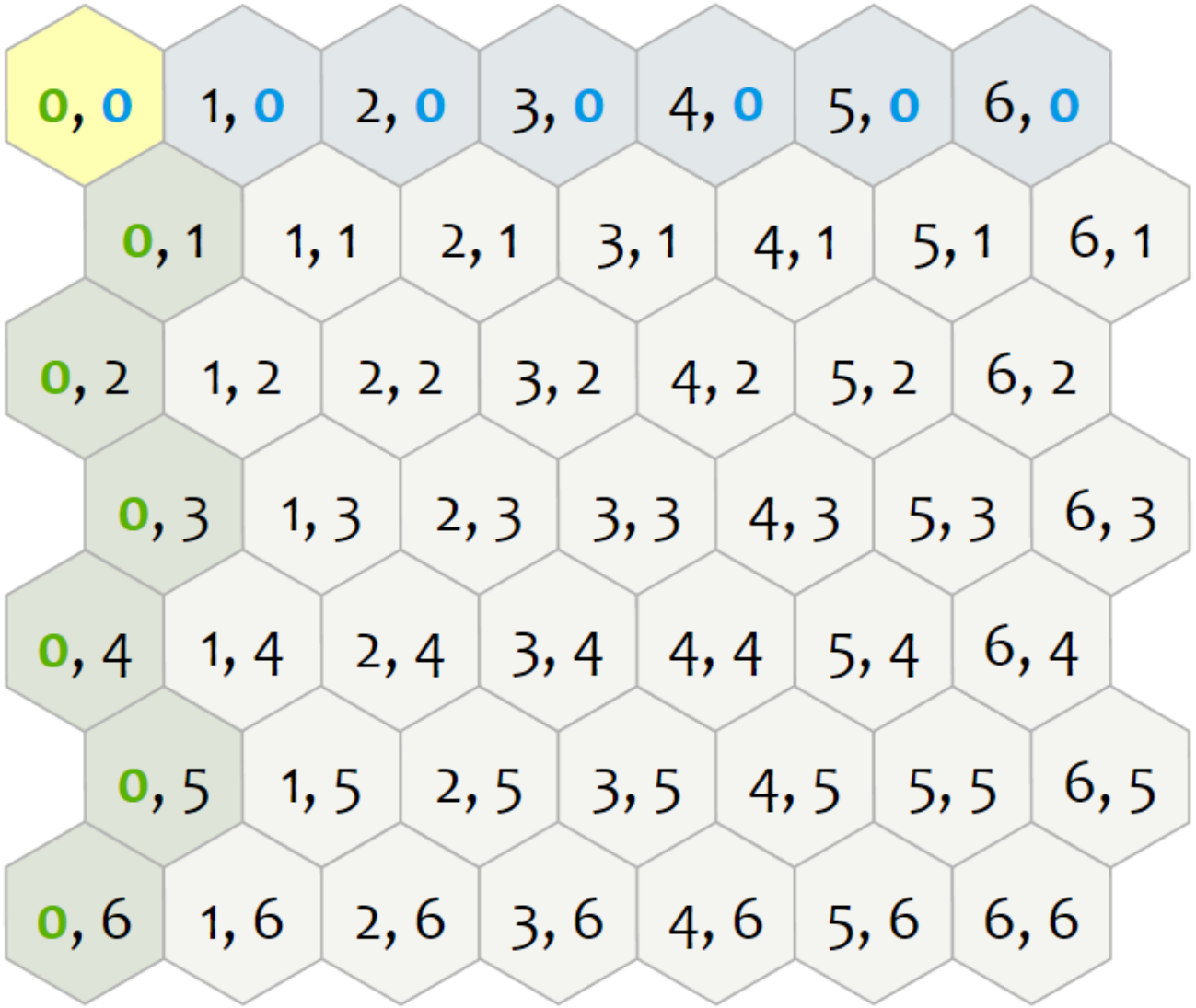


Coordenadas axiais

- As coordenadas axiais são as mesmas que as de cubo mas sem uma das coordenadas.

Coordenadas offset

- Eu optei pelas coordenadas offset por ser o tipo de coordenadas que considero mais intuitivo.



Implementação das coordenadas

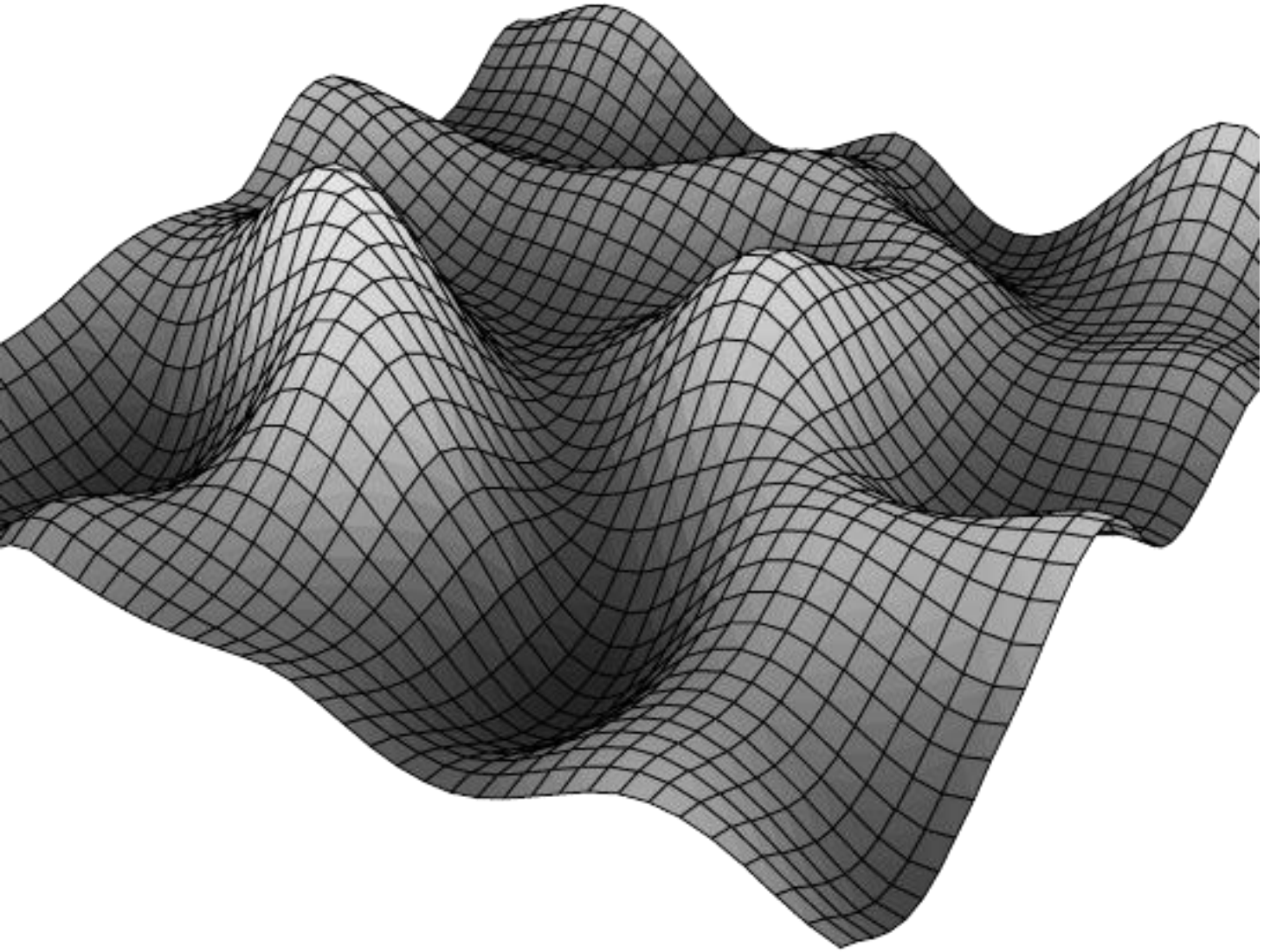
- Aqui está uma função que recebendo a posição 'verdadeira' de uma hexágono, converte-a para as coordenadas offset.

```
public static HexCoordinates FromPosition(Vector3 position)
{
    float x = position.x / (HexMetrics.innerRadius * 2f);
    float y = -x;
    float offset = position.y / (HexMetrics.outerRadius * 3f);
    x -= offset;
    y -= offset;
    int iX = Mathf.RoundToInt(x);
    int iY = Mathf.RoundToInt(-x - y);

    return new HexCoordinates(iX, iY);
}
```

Propriedades individuais de cada hexágono

- Neste jogo o jogador obtém a maior parte dos seus recursos inicias através das propriedades de cada hexágono, coisas como:
  - Recursos;
  - Tipo de terreno;
  - Alterações no terreno como florestas/oásis/colinas

```
1 reference
public bool hasWoods;
1 reference
public bool hasOasis;
1 reference
public bool hasHills;
1 reference
public bool hasStructure;
1 reference
public bool hasHorses;
1 reference
public bool hasIron;
1 reference
public bool hasNiter;
1 reference
public bool hasCoal;
1 reference
public bool hasOil;
1 reference
public bool hasAluminium;
1 reference
public bool hasUranium;
```



Criação do mapa

- Para criar um sem transições abruptas entre hexágonos utilizei o algoritmo Perlin Noise. Defini em quais intervalos criaria que tipo de hexágono, dando ao algoritmo apenas a posição do hexágono e uma seed gerada aleatoriamente.

Recursos

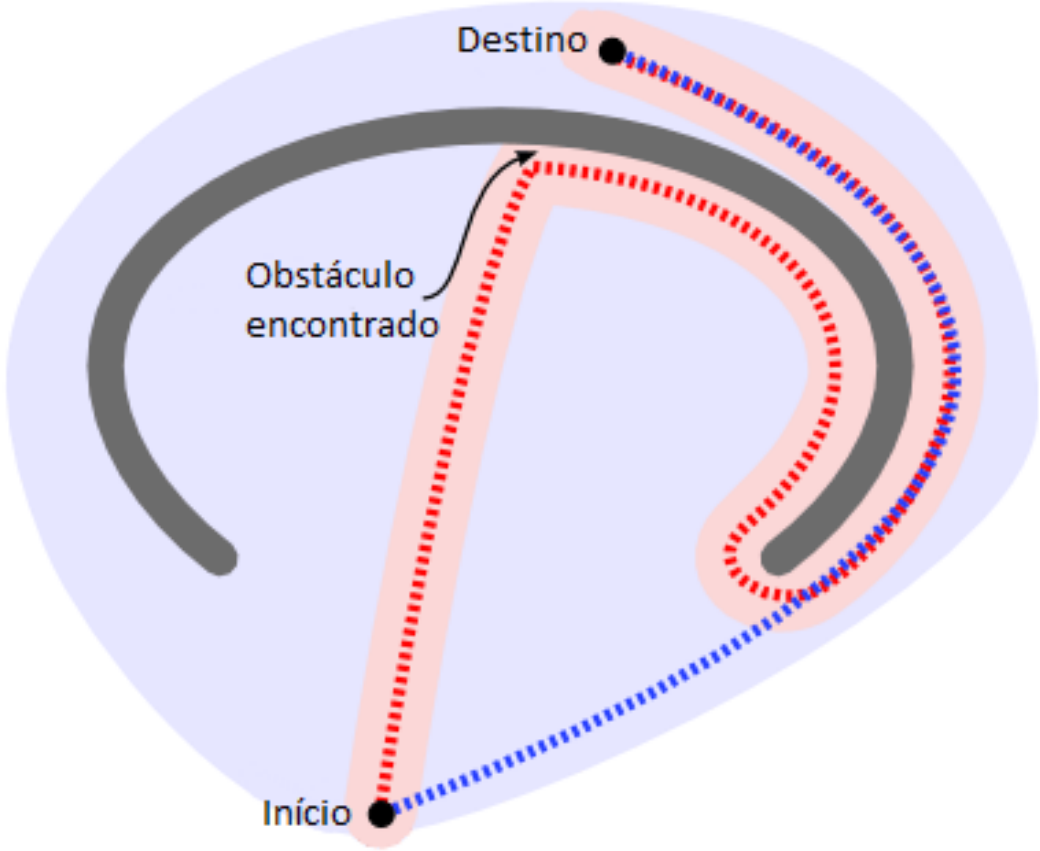
Dependendo das características de um hexágono, este poderá ter certos recursos, que influenciarão os seus rendimentos.

Rendimentos

A cada hexágono é atribuído valores de rendimentos que pode dar ao jogador, dependendo das suas características e recursos.

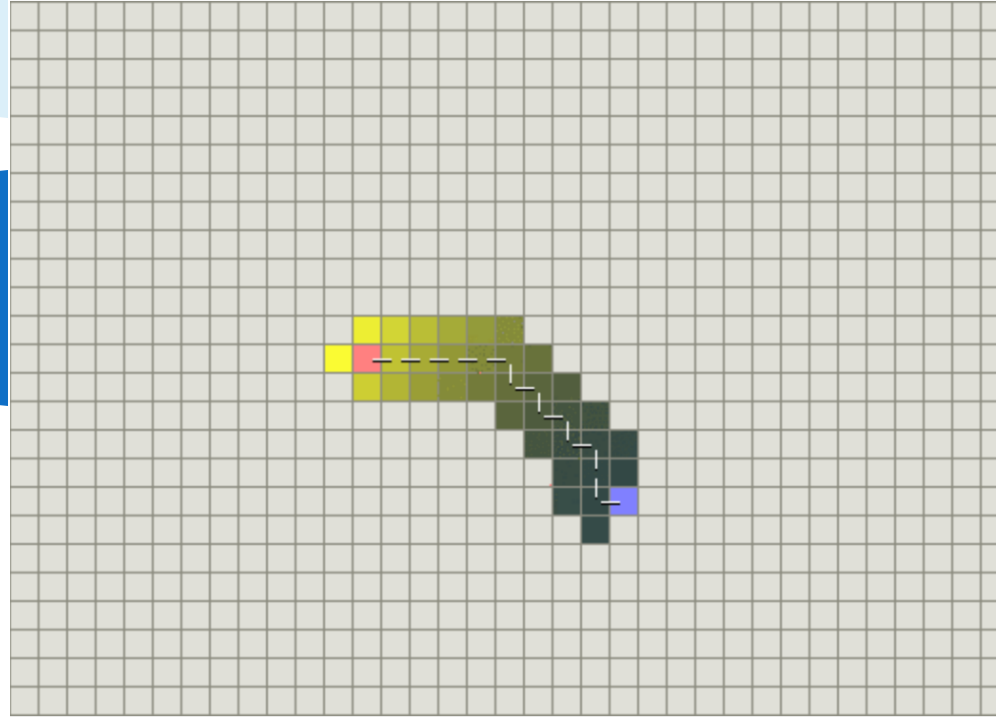
Movimento de unidades

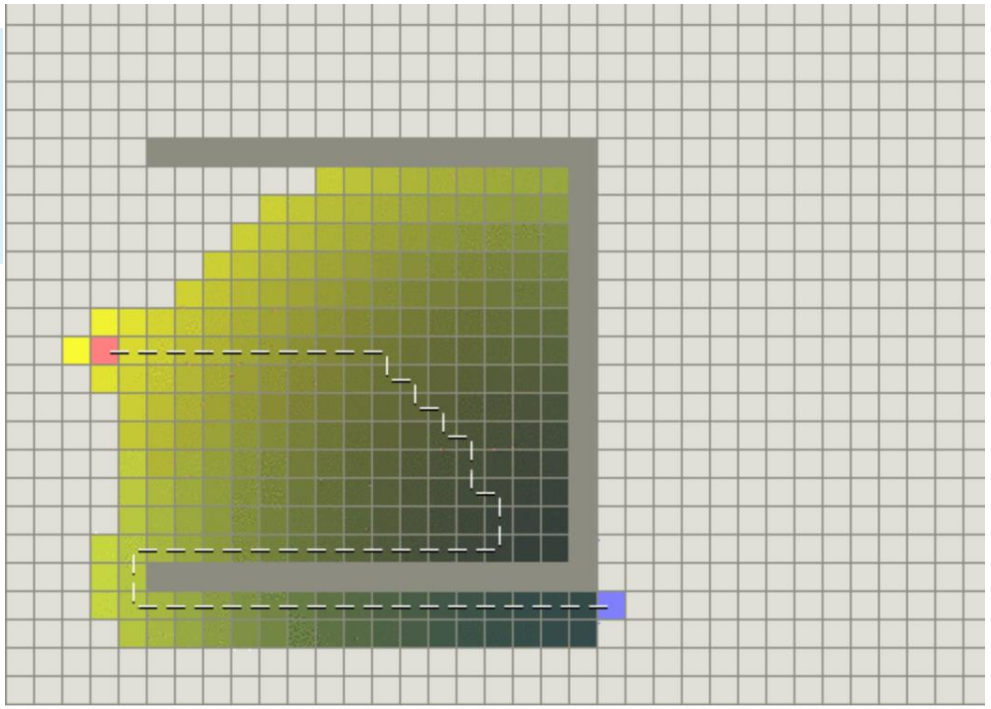
- Existem alguns algoritmos para encontrar um caminho mais curto de um ponto a outro, os mais comuns sendo:
  - BFS (Best first search);
  - Algoritmo A*;
  - Algoritmo de Dijkstra;



Algoritmo BFS

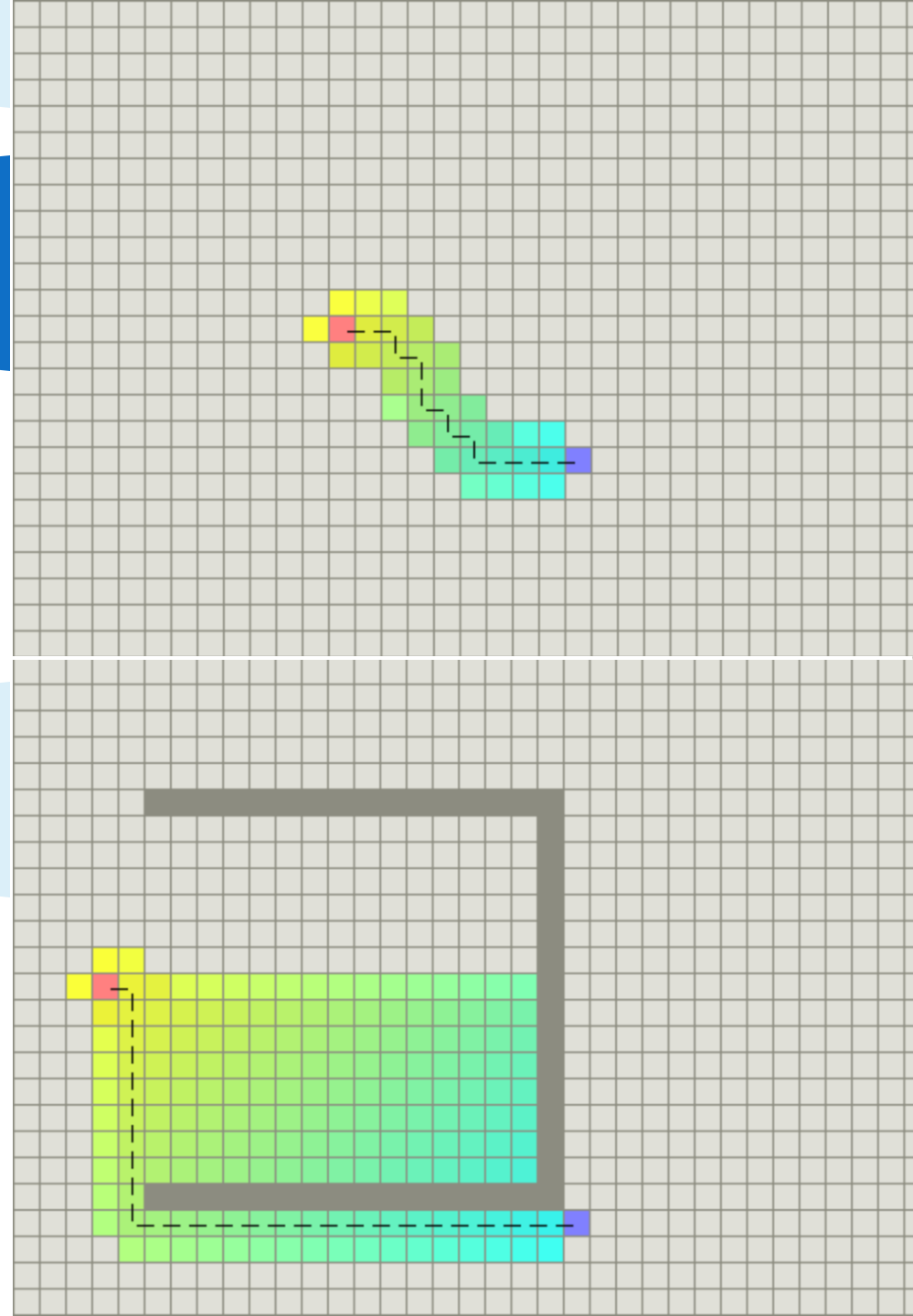
- Em condições ideais este é o mais rápido dos algoritmos, pois segue um caminho direto ao destino.

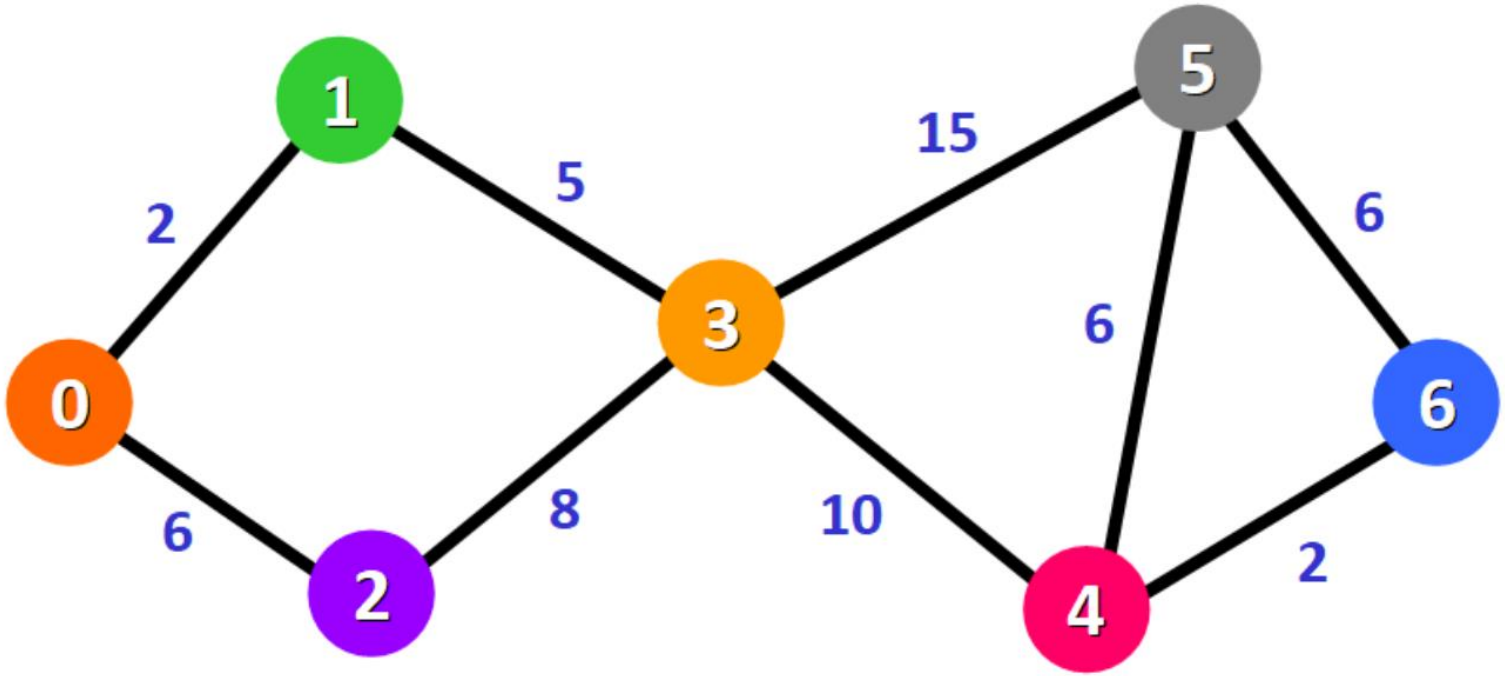




Algoritmo A*

- O algoritmo A* é o mais eficiente para encontrar um caminho mais curto em qualquer mapa.





Algoritmo de Dijkstra

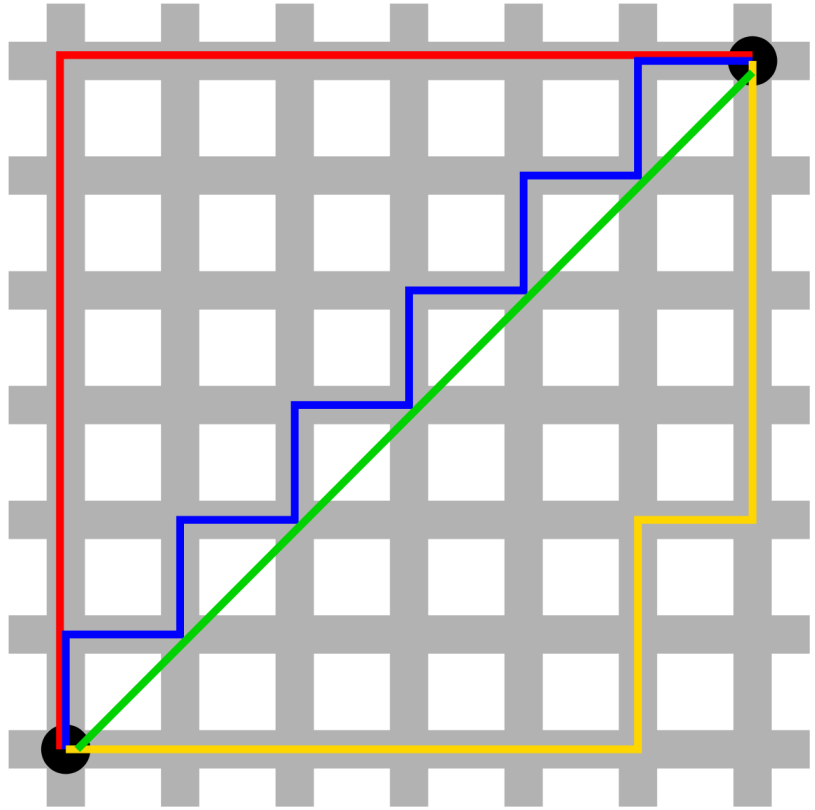
- O algoritmo de Dijkstra encontra o caminho mais curto do ponto inicial a qualquer outro ponto em um grafo com pesos.

Adjacência

- Para calcular a distância entre dois hexágonos eu utilizei uma heurística.

Heurística

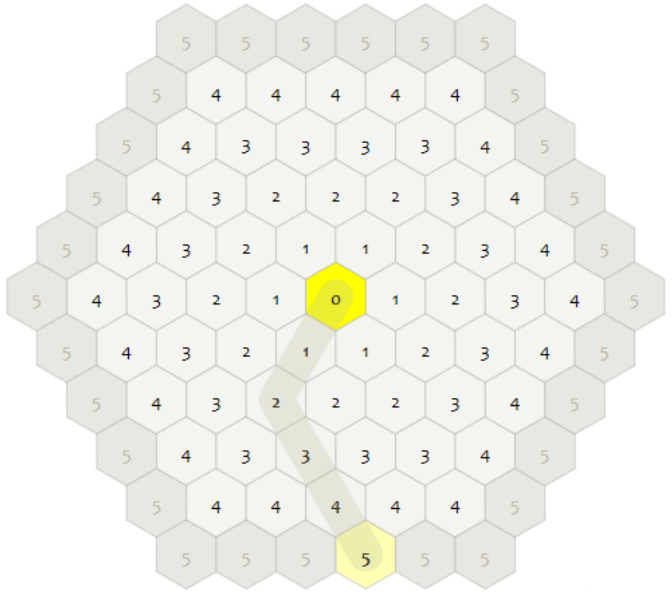
- A heurística é uma forma de calcular a quantia de 'passos' necessário para passar de um ponto a outro em uma grelha.



Conversão de coordenadas

- Tendo me conta que eu uso coordenadas offset introduzi uma função para converter estas em coordenadas de cubo.

```
public static (int q, int r, int s) OffsetToCube(HexCoordinates offset)
{
    int q = offset.x - (offset.y - (offset.y & 1)) / 2;
    int r = offset.y;
    int s = -q-r;
    return (q, r, s);
}
```




```
public static int Heuristic(HexCell S, HexCell D)
{
    int startX = S.coordinates.X;
    int startY = S.coordinates.Y;
    int endX = D.coordinates.X;
    int endY = D.coordinates.Y;
    HexCoordinates startOffset = HexCoordinates.OffsetCoordinates(startX, startY);
    HexCoordinates endOffset = HexCoordinates.OffsetCoordinates(endX, endY);
    (int startQ, int startR, int startS) = HexCoordinates.OffsetToCube(startOffset);
    (int endQ, int endR, int endS) = HexCoordinates.OffsetToCube(endOffset);
    int h = (Mathf.Abs(startQ - endQ) + Mathf.Abs(startR - endR) + Mathf.Abs(startS - endS)) / 2;
    return h;
}
```

Fórmula para heurística

- Agora que já tenho as coordenadas de cubo do hexágono inicial e final introduzo-as na seguinte função para saber a quantos hexágonos de distância estão um do outro.

Construções e unidades

O jogador pode, a partir de uma cidade, produzir construções e unidades, estas demorarão uma certa quantia de turnos para completar de acordo com os rendimentos da cidade

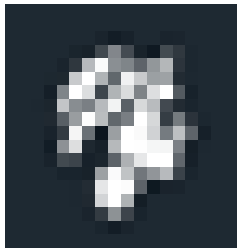
Construções

- O jogador tem a possibilidade de construir as seguintes coisas:
  - Centro de cidade;
  - Praça de teatro;
  - Universidade;
  - Polo comercial;
- Sendo que as três ultimas só podem ser construídas quando adjacentes ao um centro de cidade.

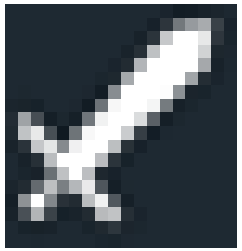
Unidades

Existem dois tipos de unidades:

- Colono;



- Guerreiro;



Colono

O colono, é a unidade que estabelece as cidades do jogador.

```
public void BuildCC()
{
    hexGrid.ChooseBuilding(this.GetComponentInParent<UnitMovement>().startingCell.coordinates.X, this.GetComponentInParent<UnitMovement>().startingCell.coordinates.Y, 1);
    foreach(GameObject btn in this.GetComponentInParent<UnitMovement>().btns)
    {
        Destroy(btn);
    }
    settlerUI.SetActive(false);
    hexGrid.settlers.Remove(this.gameObject);
    this.transform.DetachChildren();
    Destroy(this.gameObject);
}
```

Guerreiro

O guerreiro consegue atacar unidades adversárias e derrotar cidades

Cidades-Estado

Existem 4 cidades-estado no jogo, cada uma com 2 guerreiros. Estas cidades podem ser derrotadas pelo jogador.

Agora para a parte prática