**Chokepoint-Finder**  
Joe Karston Harvey Smith

# Contents

Contents

[Contents 2](#_Toc162027180)

[Analysis 3](#_Toc162027181)

[Problem Description 3](#_Toc162027182)

[Inspiration/Problem origin 3](#_Toc162027183)

[Description 3](#_Toc162027184)

[Hexagonal Grid Map 3](#_Toc162027185)

[Orientation 3](#_Toc162027186)

[Representing the Grid 3](#_Toc162027187)

[Movement/Connections 4](#_Toc162027188)

# Analysis

## Problem Description

### Inspiration/Problem origin

The problem is inspired from a game called Sid Meier’s Civilization® VI in which the game is made up of hexagonal grid and players control cities and can attack other players cities. Their troops must traverse the from one city to another. Their paths may be blocked by terrain such as mountains, lakes, coast, and deep water. While Sid Meier’s Civilization® VI is the inspiration for this project I will not be following all the details present in the game, such as defence values or embarking.

### Description

Create an algorithm to take a map (Grid of Hexagon tiles) and output the minimum number of tiles required to block all paths from an attacker city to the defender city. The map will be made up of passable and impassable tiles as well as one attacker and one defender tile.

## Hexagonal Grid Map

|  |  |
| --- | --- |
| Hexagon Orientation | |
| Horizontal | Vertical |
|  |  |

### Orientation

Sid Meier’s Civilization® VI’s map is composed of uniform hexagonal tiles in a grid. The grid is a horizontal orientation due to the pointed tops of the hexagons. This is important as it governs the movement of one tile to another.

### Representing the Grid

The gird is two-dimensional and therefore intuitively a two-dimensional array would be appropriate to store and represent the grid. The Rows are intuitively numbered however columns zig zag down due to the hexagonal shape. This poses some complexity for movement.

A white and blue hexagon pattern

Description automatically generated

### Movement/Connections

A diagram of a hexagon with arrows

Description automatically generatedOnly one troop may occupy one tile at a time, they move from one tile to another granted the two tiles share an edge and that tile is ‘passable’ this mean the tile is not a mountain, lake, coast, or deep water.

#### Finding Adjacent Tiles

If we implement the grid as a 2D array finding tiles directly to the left or right of a tile is easy as we can just increment the x index. However, finding tiles above and below is slightly more complex.

Tiles on an odd row are offset compared to an even row and therefore governs its own ruleset.

A close-up of a diagram

Description automatically generated

This rule can be accounted for by considering the tiles Y co-ordinate. If Y is an odd value, then when moving to a tile above or below add +1 to the x co-ordinate of the target tile.

|  |  |  |
| --- | --- | --- |
| Edge Tile | X | Y |
| Top left | Me.X – 1 + oddOffset | Me.Y – 1 |
| Top right | Me.X + oddOffset | Me.y – 1 |
| Bottom left | Me.X – 1 + oddOffset | Me.Y + 1 |
| Bottom right | Me.X + oddOffset | Me.y + 1 |
| Left | Me.X – 1 | Me.Y |
| Right | Me.X + 1 | Me.Y |

### Tile Attributes

Each tile is deemed either passable or impassable. Passable tiles may contain one troop and one city.

## Blocking All paths.

All paths are considered blocked when there is no route from the attacker city to the defender city comprised of only passable and unblocked tiles.

The algorithms job is to find the minimum number of tiles needed to block all paths from a an attacker city to a defender city

## Identifying Tiles to Block

### Brute force

One method to finding the minimum number of tiles needed to prevent any path between 2 nodes is by testing all possibilities and finding one with the lowest number of blocked tiles.

There are 2^n possible configurations of blocked tiles - where n is the number of tiles. After generating all possibilities, we also need to check is a solution blocks all connection between 2 nodes on a graph. This is typically performed using either a BFS or DFS. With a time, complexity of O(n+E) – where n is the number of nodes(tiles) and E is the number of edges.

This means that the problem can be solved in a time of O(2^n). Therefore, this problem cannot be solved in a tractable amount of time using brute force.

### Alternative solutions

A challenge for this project can be to try find a solution that can generate a perfect or near perfect answer is a quicker time. For example, start with all possible solution using only 1 blocked tile. If none are found move to 2 blocked tiles and so on. While this wouldn’t reduce the big O complexity it would reduce its Average case.

While a faster solution could be found by optimising the brute force method, I want to test myself and try come up with alternatives to solve the problem in a tractable amount of time.

### Chokepoint definition

#### Real world definition:

Choke points are geographic locations where the flow of people and goods can be constricted and choked off in the event of a conflict.

#### Problem definition:

Defining a chokepoint can be difficult. As easy definition is a what’s known as a bridge in graph theory. This is a single edge that when removed separates a graph into 2 separate graphs. However, this solution does not work practically for the game this problem is inspired from as it would present almost all maps as not having meaningful bridges. There is then many other ways one can try to define a chokepoint.

Areas or single tiles that when blocked:  
Prevent a large quantity or all of paths  
Drastically increase the distance from source to destination

But what we care about is finding the minimum number of tiles blocked require preventing all paths from a source to a destination.