~\OneDrive - St Paul's Catholic College\Documents\2D Strategy Game - Liberator\Assets\Scripts\Movement\OptimalPath.cs

```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class OptimalPath : MonoBehaviour
{
    public static List<HexData> optimalPath = new List<HexData>(); // collects hexes in a list for the optimal path to take
    public static HexData nextStep; // hex included in optimal path list
    public List<Image> landscapes = new List<Image>(); // collects images of hexes included in optimal path
    HexData targetHex; // targeting position, clicked hex
    IAdjacentFinder AdjacentOption = new PosPath(); // accessing the posPath script, which has the GetAdajcentHexesExtended method
    MoveSoldier move;

    // collects hexes in optimal path list and highlights them
    internal void MatchPatch()
    {
        optimalPath.Clear(); // clears the list before re filling
        targetHex = Controller.targetToMove; // first hex included in optimal path
        optimalPath.Add(targetHex);

        int steps = targetHex.distanceText.distanceFromStartingPoint; //gets the value of the distanceFromStartingPoint in DistanceText
        for(int i = steps; i > 1;) // iterates to find out all the hexes to be included in the optimal path
        {
            AdjacentOption.GetAdajacentHexesExtended(targetHex); // finds out hexes adjacent to targethex
            targetHex = nextStep; // when the hex is included in list it becomes a new target hex
            i -= nextStep.distanceText.MakeMePartOfOptimalPath(); // decreases the i variable by stepsToGo value

        }
        ManagePath();
    }

    // Start is called before the first frame update
    void Start()
    {
        move = GetComponent<MoveSoldier>();
    }


    void ManagePath() // reveres the optimal path, fills the path with images of the hexes
    {
        landscapes.Clear(); // clears the list before re-filling
        optimalPath.Reverse(); // sorts list elements in the opposite orders
        foreach(HexData hex in optimalPath)
        {
            landscapes.Add(hex.Landscape); // fills the list with images

        }
        move.path = landscapes; // sends information regarding the optimal path to the Move class
    }
}

```