

Michael Boone
SID: 992186566
AI project 1 ecs 170

Part 3:
java Main DijkstraAI AStarExp_992186566 -seed 1
DijkstraAI
Path Cost: 533.4482191461119
Uncovered: 226916
Time taken: 2712

AStarExp_992186566
Path Cost: 533.4482191461119
Uncovered: 71816
Time taken: 468

java Main DijkstraAI AStarExp_992186566 -seed 2
DijkstraAI
Path Cost: 549.5036346739352
Uncovered: 237620
Time taken: 2854

AStarExp_992186566
Path Cost: 549.5036346739352
Uncovered: 81987
Time taken: 520

java Main DijkstraAI AStarExp_992186566 -seed 3
DijkstraAI
Path Cost: 510.97825243663607
Uncovered: 228705
Time taken: 2715

AStarExp_992186566
Path Cost: 510.97825243663607
Uncovered: 74085
Time taken: 481

java Main DijkstraAI AStarExp_992186566 -seed 4
DijkstraAI
Path Cost: 560.6570436319696
Uncovered: 216096
Time taken: 2484

AStarExp_992186566
Path Cost: 560.6570436319696
Uncovered: 66491
Time taken: 426

java Main DijkstraAI AStarExp_992186566 -seed 5
DijkstraAI
Path Cost: 479.5879215923168
Uncovered: 220673
Time taken: 2461

AStarExp_992186566
Path Cost: 479.5879215923168
Uncovered: 67904
Time taken: 459

java Main DijkstraAI AStarDiv_992186566 -seed 1
DijkstraAI
Path Cost: 39.0
Uncovered: 235697
Time taken: 1455

AStarDiv_992186566
Path Cost: 39.0
Uncovered: 238628
Time taken: 498

java Main DijkstraAI AStarDiv_992186566 -seed 2
DijkstraAI
Path Cost: 40.0
Uncovered: 244248
Time taken: 1437

AStarDiv_992186566
Path Cost: 40.0
Uncovered: 243894
Time taken: 525

java Main DijkstraAI AStarDiv_992186566 -seed 3
DijkstraAI
Path Cost: 37.0
Uncovered: 231995
Time taken: 1432

AStarDiv_992186566
Path Cost: 37.0
Uncovered: 234596
Time taken: 558

java Main DijkstraAI AStarDiv_992186566 -seed 4
DijkstraAI
Path Cost: 41.0
Uncovered: 237889
Time taken: 1457

AStarDiv_992186566

Path Cost: 41.0

Uncovered: 239141

Time taken: 500

java Main DijkstraAI AStarDiv_992186566 -seed 5

DijkstraAI

Path Cost: 35.0

Uncovered: 223768

Time taken: 1332

AStarDiv_992186566

Path Cost: 35.0

Uncovered: 227592

Time taken: 478

part 1:

EXP COST FUNCTION = $e^{(p2-p1)}$

- if $p1 = p2$ upper bound is 1
- if $p1 < p2$ upper bound is e^{255}
- if $p1 > p2$ upper bound is e^{-1}

Exp Cardinal Movement:

Cardinal Movement Min is = $\text{abs}(pt1.x - pt2.x) + \text{abs}(pt1.y - pt2.y)$

let this be manhattan;

let heightDif be height of $pt1$ – height of $pt2$;

let absHeightDif be $\text{abs}(\text{heightDif})$;

```
private double getHeuristic(final TerrainMap map, final Point pt1, final Point pt2) //e^(f2-f1)
{
```

```
    double chebyshev = Math.max(Math.abs(pt1.x - pt2.x), Math.abs(pt1.y - pt2.y));
```

```
    double p1H = (double)map.getTile(pt1);
```

```
    double p2H = (double)map.getTile(pt2);
```

```
    double heightDifference = p1H - p2H;
```

```
    double currentHeight;
```

```
    double absoluteHeightDifference = Math.abs(heightDifference);
```

```
    double stepWidth;
```

```
    double estCost;
```

```
    if(heightDifference == 0)
```

```
    {
```

```
        return manhattan;
```

```
    }
```

```

else if(heightDifference > 0)//e^(-1) is max p1 above p2
{
    double averageMovement = (absoluteHeightDifference/manhattan);
    currentHeight = p1H - averageMovement;
    estCost = Math.exp(-averageMovement);
    if(averageMovement < 1)
    {
        estCost = 0;
        for(int i = 0; i <= absoluteHeightDifference; i++)
        {
            estCost += Math.exp((double)-1.0);
        }
        estCost += (manhattan - absoluteHeightDifference);
        return Math.floor(estCost);
    }

    while(currentHeight > p2H)
    {
        averageMovement = Math.ceil(averageMovement);
        estCost += Math.exp(-averageMovement);
        currentHeight = currentHeight - averageMovement;
    }

    return Math.floor(estCost);
}
else//(heightDifference < 0)//e^255 is max p1 below p2
{
    double averageMovement = (absoluteHeightDifference/manhattan);
    currentHeight = p1H - averageMovement;
    estCost = Math.exp(averageMovement);

    if(averageMovement <= 1)
    {
        estCost = 0;
        for(int i = 0; i < absoluteHeightDifference; i++)
        {
            estCost += Math.exp((double)1.0);
        }
        estCost += (manhattan - absoluteHeightDifference);
        return Math.floor(estCost);
    }

    while(currentHeight < p2H)
    {
        averageMovement = Math.floor(averageMovement);
        estCost += Math.exp(averageMovement);
        currentHeight = currentHeight + averageMovement;
    }
}

```

```

        return Math.floor(estCost);
    }
}

```

Proof of Admissability:

case 1: $p1.h = p2.h$

assume $h(n) > h^*$

then $e^{(p1.h - p2.h)} > e^{(p1.h - p2.h)} + \text{Integral other nodes}$

but! $1 > 1 + \text{integral other nodes}$

this cant happen! Other nodes would have to all return 0 and there is no case where e^x returns 0!

case 2: $p1.h < p2.h \ \&\& \ \text{abs}(\text{slope}) \leq 1$

assume $h(n) > h^*$

then $e^{(1.0)*\text{heightDifference}} + (\text{manhattan} - \text{heightDifference}) > e^{(1.0)*\text{heightDifference}} + \text{integral of rest};$

then $e^{(\text{heightDifference})} + (\text{rest of nodes} * 1) > e^{(\text{heightDifference})} + \text{integral of rest};$

then $\text{rest of nodes} * 1 > \text{integral of rest};$

but! For this to be valid there must be a case such that $e^{-x} + e^x < 1$ this can not happen

case 3: $p1.h > p2.h \ \&\& \ \text{abs}(\text{slope}) \leq 1$

assume $h(n) > h^*$

then $e^{(-1.0)*\text{heightDifference}} + (\text{manhattan} - \text{heightDifference}) > e^{(-1.0)*\text{heightDifference}} + \text{integral of rest};$

rest;

then $(\text{rest of nodes} * 1) > \text{integral of rest};$

but! For this to be valid there must be a case such that $e^{-x} + e^x < 1$ this can not happen

case 4: $p1.h < p2.h \ \&\& \ \text{abs}(\text{slope}) > 1$

assume $h(n) > h^*$

integral $e^{(\text{abs}(\text{slope})/\text{manhattan})} > \text{integral of rest of nodes}$

for this to happen there has to be a case where:

$$e^{(\text{abs}(\text{slope})/\text{manhattan})} + e^{(\text{abs}(\text{slope})/\text{manhattan})} > e^{(\text{abs}(\text{slope})/\text{manhattan}) + \delta} + e^{(\text{abs}(\text{slope})/\text{manhattan}) - \delta};$$

But! This can happen!

case 5: $p1.h > p2.h \ \&\& \ \text{abs}(\text{slope}) > 1$

assume $h(n) > h^*$

integral $e^{(-\text{abs}(\text{slope})/\text{manhattan})} > \text{integral of rest of nodes}$

for this to happen there has to be a case where:

$$e^{(-\text{abs}(\text{slope})/\text{manhattan})} + e^{(-\text{abs}(\text{slope})/\text{manhattan})} > e^{(-\text{abs}(\text{slope})/\text{manhattan}) + \delta} + e^{(-\text{abs}(\text{slope})/\text{manhattan}) - \delta};$$

$$e^{-(\text{abs}(\text{slope})/\text{manhattan}) - \text{delta}});$$

But! This can happen!

Exp 8 degree Movement:

```
private double getHeuristic(final TerrainMap map, final Point pt1, final Point pt2) //e^(f2-f1)
{
    double chebyshev = Math.max(Math.abs(pt1.x - pt2.x), Math.abs(pt1.y - pt2.y));
    double p1H = (double)map.getTile(pt1);
    double p2H = (double)map.getTile(pt2);
    double heightDifference = p1H - p2H;
    double currentHeight;
    double absoluteHeightDifference = Math.abs(heightDifference);
    double stepWidth;
    double estCost;

    if(heightDifference == 0)
    {
        return chebyshev;
    }
    else if(heightDifference > 0)//e^(-1) is max p1 above p2
    {
        double averageMovement = (absoluteHeightDifference/chebyshev);
        currentHeight = p1H - averageMovement;
        estCost = Math.exp(-averageMovement);
        if(averageMovement < 1)
        {
            estCost = 0;
            for(int i = 0; i <= absoluteHeightDifference; i++)
            {
                estCost += Math.exp((double)-1.0);
            }
            estCost += (chebyshev - absoluteHeightDifference);
            return Math.floor(estCost);
        }

        while(currentHeight > p2H)
        {
            averageMovement = Math.ceil(averageMovement);
            estCost += Math.exp(-averageMovement);
            currentHeight = currentHeight - averageMovement;
        }

        return Math.floor(estCost);
    }
    else//(heightDifference < 0)//e^255 is max p1 below p2
    {
        double averageMovement = (absoluteHeightDifference/chebyshev);
```

```

currentHeight = p1H - averageMovement;
estCost = Math.exp(averageMovement);

if(averageMovement <= 1)
{
    estCost = 0;
    for(int i = 0; i < absoluteHeightDifference; i++)
    {
        estCost += Math.exp((double)1.0);
    }
    estCost += (chebyshev - absoluteHeightDifference);
    return Math.floor(estCost);
}

while(currentHeight < p2H)
{
    averageMovement = Math.floor(averageMovement);
    estCost += Math.exp(averageMovement);
    currentHeight = currentHeight + averageMovement;
}

return Math.floor(estCost);
}
}

```

Proof of Admissability:

case 1: $p1.h = p2.h$

assume $h(n) > h^*$

then $e^{(p1.h - p2.h)} > e^{(p1.h - p2.h)} + \text{Integral other nodes}$

but! $1 > 1 + \text{integral other nodes}$

this cant happen! Other nodes would have to all return 0 and there is no case where e^x returns 0!

case 2: $p1.h < p2.h \ \&\& \ \text{abs(slope)} \leq 1$

assume $h(n) > h^*$

then $e^{(1.0)*\text{heightDifference}} + (\text{chebyshev} - \text{heightDifference}) > e^{(1.0)*\text{heightDifference}} + \text{integral of rest};$

then $e(\text{heightDifference}) + (\text{rest of nodes} * 1) > e(\text{heightDifference}) + \text{integral of rest};$

then $\text{rest of nodes} * 1 > \text{integral of rest};$

but! For this to be valid there must be a case such that $e^{-x} + e^x < 1$ this can not happen

case 3: $p1.h > p2.h \ \&\& \ \text{abs(slope)} \leq 1$

assume $h(n) > h^*$

then $e^{(-1.0)*\text{heightDifference}} + (\text{chebyshev} - \text{heightDifference}) > e^{(-1.0)*\text{heightDifference}} + \text{integral of rest};$

then $(\text{rest of nodes} * 1) > \text{integral of rest};$

but! For this to be valid there must be a case such that $e^{-x} + e^x < 1$ this can not happen

case 4: $p1.h < p2.h \ \&\& \ \text{abs}(\text{slope}) > 1$

assume $h(n) > h^*$

$\text{integral } e^{(\text{abs}(\text{slope})/\text{chebyshev})} > \text{integral of rest of nodes}$

for this to happen there has to be a case where:

$$e^{(\text{abs}(\text{slope})/\text{chebyshev})} + e^{(\text{abs}(\text{slope})/\text{chebyshev})} > e^{(\text{abs}(\text{slope})/\text{chebyshev}) + \delta} + e^{(\text{abs}(\text{slope})/\text{chebyshev}) - \delta};$$

But! This can happen!

case 5: $p1.h > p2.h \ \&\& \ \text{abs}(\text{slope}) > 1$

assume $h(n) > h^*$

$\text{integral } e^{(-\text{abs}(\text{slope})/\text{chebyshev})} > \text{integral of rest of nodes}$

for this to happen there has to be a case where:

$$e^{(-\text{abs}(\text{slope})/\text{chebyshev})} + e^{(-\text{abs}(\text{slope})/\text{chebyshev})} > e^{(-(\text{abs}(\text{slope})/\text{chebyshev}) + \delta)} + e^{(-(\text{abs}(\text{slope})/\text{chebyshev}) - \delta)};$$

But! This can happen!

DIV COST FUNCTION = $p2/(p1+1) \rightarrow$ integer math

* this leads to if $p1$ is higher than $p2$ cost is 0, if $p1 = p2$ cost is 0;

* if $p1$ is lower than $p2$ the cost goes between 1 and 255 depending on the width between them.

Div Cardinal Movement:

Cardinal Movement Min is = $\text{abs}(pt1.x - pt2.x) + \text{abs}(pt1.y - pt2.y)$

let this be manhattan;

```
/*private double getHeuristic(final TerrainMap map, final Point pt1, final Point pt2) //
{
    double p1H = (double)map.getTile(pt1);
    double p2H = (double)map.getTile(pt2);
    double chebyshev = Math.max(Math.abs(pt1.x - pt2.x), Math.abs(pt1.y - pt2.y));
    double heightDifference = p1H - p2H;
    double absoluteHeightDifference = Math.abs(heightDifference);
    double chunkWidth;
    double total = 0;
    double tempH;

    if(heightDifference > 0) // p1 is above p2
    {
        return 0;
    }
    else // p1 is below p2
    {
        return 1;
    }
}*/
```


Proof of Admissability:

case 1: $p1 \geq p2$

the cost function returns 0 for this case so return 0

assume $h(n) > h^*$

$0 > 0?$

case 2: $p1 < p2$

assume $h(n) > h^*$

$1 > \text{integral of all other nodes}$

but, the integral of the next node must at least be 1 height difference which would return 1

so $1 > 1 + \text{integral of rest of nodes}$

This cant happen!

Div 8 degree Movement:

```
/*private double getHeuristic(final TerrainMap map, final Point pt1, final Point pt2) //
{
    double p1H = (double)map.getTile(pt1);
    double p2H = (double)map.getTile(pt2);
    double chebyshev = Math.max(Math.abs(pt1.x - pt2.x), Math.abs(pt1.y - pt2.y));
    double heightDifference = p1H - p2H;
    double absoluteHeightDifference = Math.abs(heightDifference);
    double chunkWidth;
    double total = 0;
    double tempH;

    if(heightDifference > 0) // p1 is above p2
    {
        return 0;
    }
    else // p1 is below p2
    {
        return 1;
    }
}*/
```

Proof of Admissability:

case 1: $p1 \geq p2$

the cost function returns 0 for this case so return 0

assume $h(n) > h^*$

$0 > 0?$

case 2: $p1 < p2$

assume $h(n) > h^*$

$1 > \text{integral of all other nodes}$

but, the integral of the next node must at least be 1 height difference which would return 1

so $1 > 1 + \text{integral of rest of nodes}$

This can't happen!

Part 4:

a) I didn't change A^* compared to the rest.

My heuristic was consistent so I removed the closed list.

I then made my open list a minHeap tree so sorting was $\log(n)$ and it auto sorted.

```
java Main DijkstraAI MtStHelensExp_992186566 -load MTAFT.XYZ
```

DijkstraAI

Path Cost: 515.6645805015318

Uncovered: 1203049

Time taken: 13739

MtStHelensExp_992186566

Path Cost: 515.6645805015318

Uncovered: 119297

Time taken: 1352