Report A4

Vicky Prakash (Net Id: vp407), Whitney Poh, John Fuller

A star:

YouTube Link: https://www.youtube.com/watch?v=GXlpJpcrKqI&feature=youtu.be&hd=1

Description:

- Time taken: 0.086 sec
- Nodes Expanded: 361
- Nodes generated: 454
- Path Length: 26
- Path Cost:29.97

Weighted A*: (weight = 5)

YouTube Link: https://www.youtube.com/watch?v=ozGrMQYis0Q&feature=youtu.be&hd=1

Description:

- Time taken: 0.045sec
- Nodes Expanded: 151
- Nodes generated: 227
- Path Length: 34
- Path Cost:37.14

ARA:

Case 1:

(Initial Weight: 1000 Decrement Rate: 1 Time Limit: 1 sec)

YouTube Link: https://www.youtube.com/watch?v=C-ILQWJIto4&feature=youtu.be&hd=1

Description:

- Path Length: 31
- Path Cost:33.07

Case 2:

(Initial Weight: 500 Decrement Rate:5 Time Limit: 2 sec)

YouTube Link: https://www.youtube.com/watch?v=UMoBu4pkEIE&feature=youtu.be&hd=1

Description:

- Path Length: 31
- Path Cost:37.14

Case 3:

(Initial Weight: 300 Decrement Rate: 3 Time Limit: 2)

YouTube Link: https://www.youtube.com/watch?v=Or5TCfbDEcI&feature=youtu.be&hd=1

Description:

- Path Length: 31
- Path Cost:37.14

Extra Credit:

Min Binary Heap:

I have implemented open_list using BinaryMinHeap. For this, I have created a class called Astar_minheap. The primary function to maintain min heap is maintain_minheap, which is called every time during insert, update, pop and remove_node function calls. In this function, I am swapping a node with its parent till the time, its parents are not larger than the node. In this manner, we maintain the root to be the node with the smallest f-value. And in case of tie, the node with the smallest g-value is chosen (implemented using the function less).

```
// return true when first node has smaller cost thatn the second node
static bool less(double fvalue1, double gvalue1, double fvalue2, double gvalue2)
{
    if (fvalue1 < fvalue2)
        return true;
    else if (fvalue1 > fvalue2)
        return false;
    else
        return (gvalue1 < gvalue2);
}
```

```
int maintain_minheap(int node)
{
    if (node >= indexnumber_heap.size())
        return -1;
    int temp = node;
    int n = fvalue_heap.size();

    while (temp > 1 && less(fvalue_heap[temp], gvalue_heap[temp], fvalue_heap[temp >> 1], gvalue_heap[temp >> 1]))
    {
        swap(temp, (temp >> 1));
        temp = (temp >> 1);
    }

    while ((temp << 1) < n)
    {
        int left_child = (temp << 1);
        int right_child = 1 + (temp << 1);
        int best_child;
        if (right_child >= n || less(fvalue_heap[left_child], gvalue_heap[left_child], fvalue_heap[right_child], gvalue_heap[right_child]))
            best_child = left_child;
        else
            best_child = right_child;
        if (less(fvalue_heap[temp], gvalue_heap[temp], fvalue_heap[best_child], gvalue_heap[best_child]))
            break;
        swap(temp, best_child);
        temp = best_child;
    }
    return temp;
}
```

Implementation is in the header file.

Sequential A star:

YouTube Link: https://www.youtube.com/watch?v=5N74EUGyoAo&feature=youtu.be&hd=1

Description:

- Time taken: 0.1241sec
- Nodes Expanded: 503
- Nodes generated: 657
- Path Length: 26
- Path Cost:29.97

The two heuristics I have implemented are : Manhattan and Euclidean (functions euclidean_distance and manhattan_distance). However, for the above videos, I have only used Euclidean distance.

```
double euclidean_distance(Util::Point &current, Util::Point &goal, SteerLib::SpatialDataBaseInterface * _gSpatialDatabase)
{
    double dx = fabs(current.x - goal.x), dz = fabs(current.z - goal.z);
    return sqrt(dx * dx + dz * dz);
}

double manhattan_distance(Util::Point &current, Util::Point &goal, SteerLib::SpatialDataBaseInterface * _gSpatialDatabase)
{
    double dx = fabs(current.x - goal.x), dz = fabs(current.z - goal.z);
    return dx + dz;
}
```