

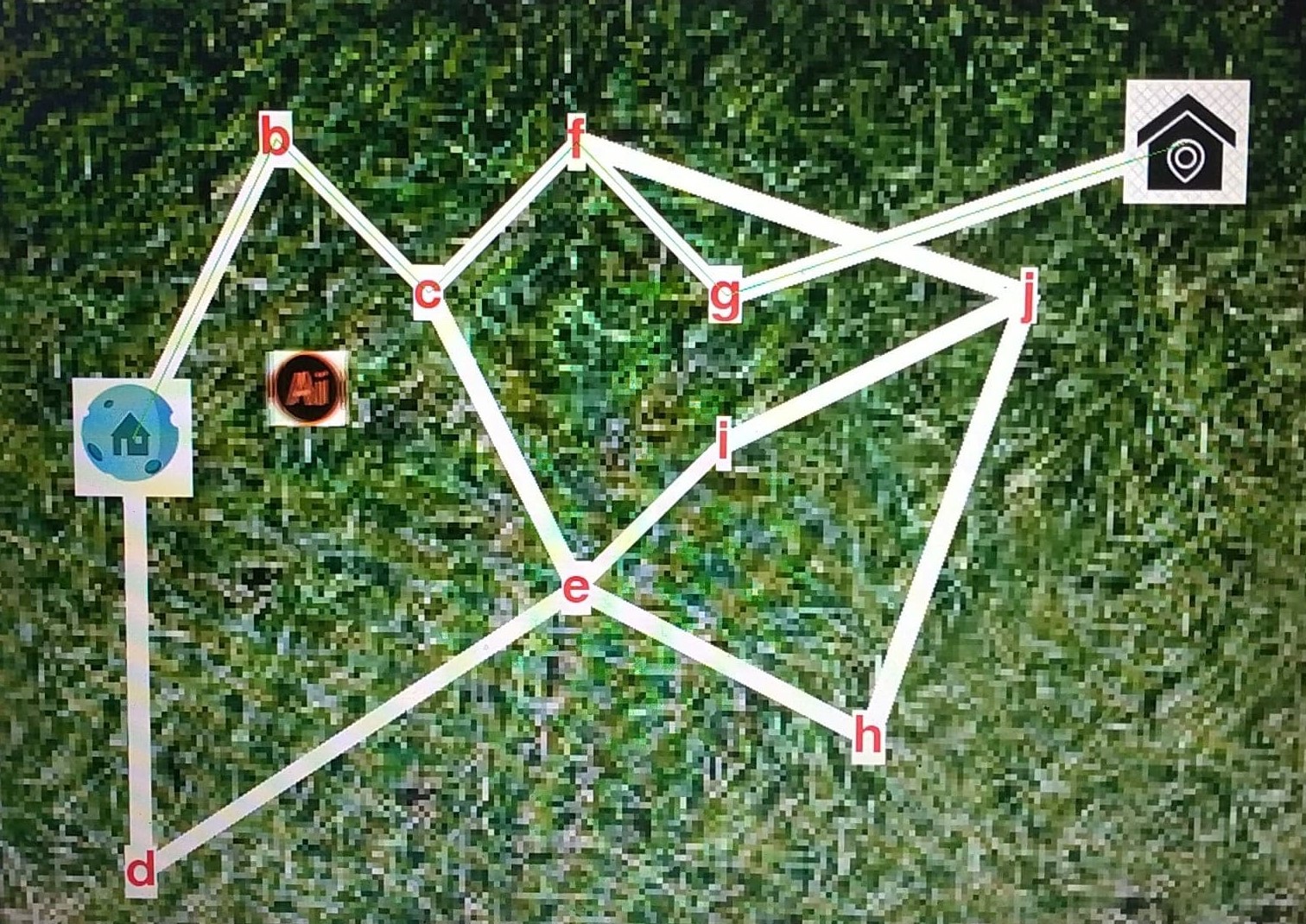
1.Abstract:-

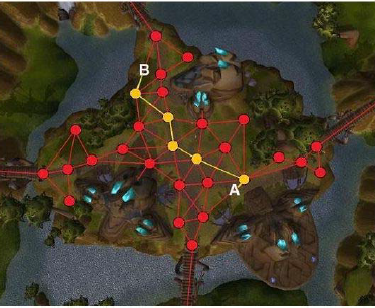
This is an artificial intelligence based game .The aim of this game is to find out the required path to attack the opponent house. It’s easier to play and very interesting for user. It’s a two player game where one player is a human and another player is computer or artificial agent.

Functionality of this game is that who can firstly reach to opponent house .So it’s also a challenging game for user cause if he or she want to win this game he or she must be reached to the opponent house before computer has been reached his/her own house.

2.Game Interface:-

Basic Path





3.User Manual:-

For success of every games make user manual is the very important task for developer team.so our team try to make a user friendly guideline so that any user can understand playing rules and functionality of this game and they also feel comport during playing.

On the previous section we said that it’s a two player game where one is human and another is computer.

This user manual is divided into two section for two type’s player

For Human:-

Thinking of every human is different so every person will decide how they play this game .But here we will mention that how they will take action to move one city to another city.

When a user click on a city by mouse or move one city to another city by arrow key of keyboard and press enter then his position moved. But selected city must have the direct path from his previous city and must be the next city of the previous city otherwise he can’t move. By repeating this step any user can reach to the goal or opponent house.

For Computer:-

Computer or AI agent will be play this game by the calculating the shortest value of the path. We know that an artificial intelligence agent perceive the environment by sensor and thinking about the next step to move and finally will take an action by his actuators. The environment of this game is fully observable for AI agent.

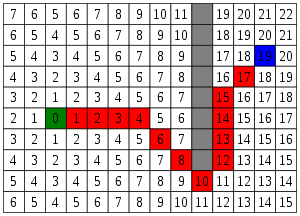
4.Implementation:-

It is a path finding problem. Where we calculate the shortest path to reach the goal node form the start node.

# F = G + H

One important aspect of A\* is f = g + h. The f, g, and h variables are in our Node class and get calculated every time we create a new node. Quickly I’ll go over what these variables mean.

* F is the total cost of the node.
* G is the distance between the current node and the start node.
* H is the heuristic — estimated distance from the current node to the end node.
* Let’s take a look at a quick graphic to help illustrate this.



Awesome! Let’s say node(0) is our starting position and node(19) is our end position. Let’s also say that our current node is at the the red square node(4).

## G

*G is the distance between the current node and the start node.*

If we count back we can see that node(4) is 4 spaces away from our start node.

We can also say that G is 1 more than our parent node (node(3)). So in this case for node(4), currentNode.g = 4.

## H

*H is the heuristic — estimated distance from the current node to the end node.*

So let’s calculate the distance. If we take a look we’ll see that if we go over 7 spaces and up 3 spaces, we’ve reached our end node (node(19)).

Let’s apply the Pythagorean Theorem! a² + b² = c². After we’ve applied this, we’ll see that currentNode.h = 7² + 3². Or currentNode.h = 58.

But don’t we have to apply the square root to 58? Nope! We can skip that calculation on every node and still get the same output. Clever!

With a heuristic, we need to make sure that we can actually calculate it. It’s also very important that the heuristic is always an underestimation of the total path, as an overestimation will lead to A\* searching for through nodes that may not be the ‘best’ in terms of f value.

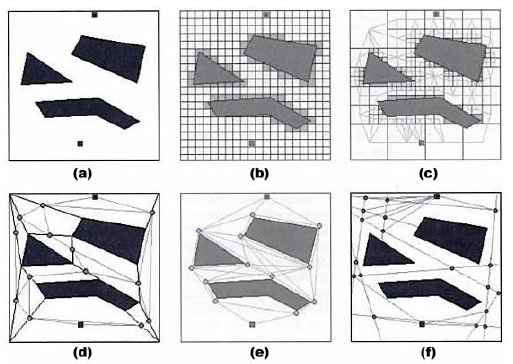
F

*F is the total cost of the node.*

So let’s add up h and g to get the total cost of our node. currentNode.f = currentNode.g + currentNode.h. Or currentNode.f = 4+ 58. Or currentNode.f = 62.

# Time to use **f** = g + h

Alright, so that was a lot of work. Now with all that work, what am I going to use this f value for?

 Fig:-search spaces

With this new f value, we can look at all our nodes and say, “Hey, is this the best node I can pick to move forward with right now?”. Rather than running through every node, we can pick the ones that have the best chance of getting us to our goal.

5.Tools:-

1. Python 3.7

2. Pygame

3. Visual studio

4. Python’s Virtual Environment Set Up

astroid==2.3.3

colorama==0.4.3

isort==4.3.21

lazy-object-proxy==1.4.3

mccabe==0.6.1

pygame==1.9.6

pylint==2.4.4

six==1.14.0

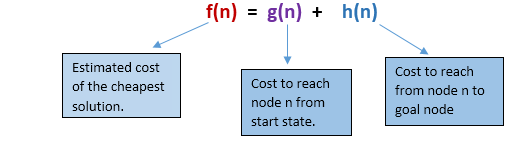
typed-ast==1.4.1

wrapt==1.11.2

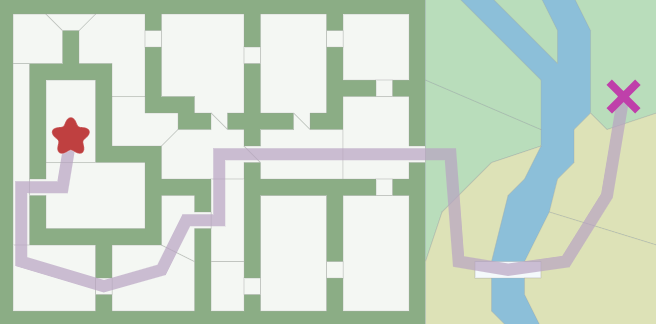
5. Adobe Illustrator

6.Algorithm:-

Name of Algorithm:- A\* Search



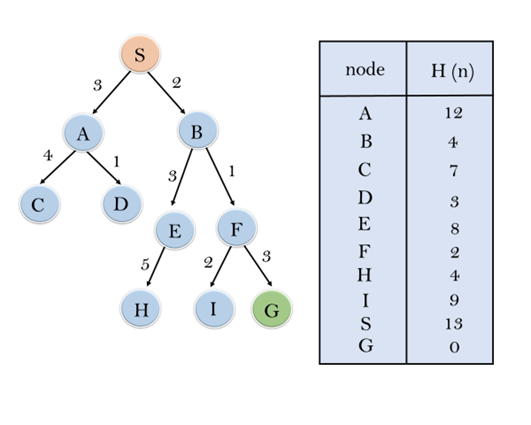
Description:- In games we often want to find paths from one location to another. We’re not only trying to find the shortest distance; we also want to take into account travel time. Move the blob  (start point) and cross  (end point) to see the shortest path.



Representing Map:-

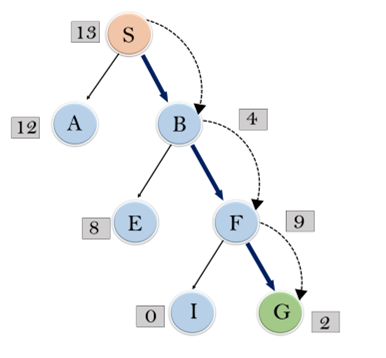
The first thing to do when studying an algorithm is to understand the **data**. What is the input? What is the output?

**Input:** Graph search algorithms, including A\*, take a “graph” as input. A graph is a set of locations (“nodes”) and the connections (“edges”) between them. Here’s the graph I gave to A\*:



A\* doesn’t see anything else. It only sees the graph. It doesn’t know whether something is indoors or outdoors, or if it’s a room or a doorway, or how big an area is. *It only sees the graph!* It doesn’t know the difference between this map and this other one.

**Output:** The path found by A\* is made of graph nodes and edges. The edges are abstract mathematical concepts. A\* will tell you to move from one location to another *but it won’t tell you how*. Remember that it doesn’t know anything about rooms or doors; all it sees is the graph. You’ll have to decide whether a graph edge returned by A\* means moving from tile to tile or walking in a straight line or opening a door or swimming or running along a curved path.



Summary **:** For any given game map, there are many different ways of making a path finding graph to give to A\*. The above map makes most doorways into nodes; what if we made doorways into edges? What if we used a path finding grid?

The path finding graph doesn’t have to be the same as what your game map uses. A grid game map can use a non-grid path finding graph, or vice versa. A\* runs fastest with the fewest graph nodes; grids are often easier to work with but result in lots of nodes. This page covers the A\* algorithm but not graph design.

**A\*** is a modification of Dijkstra’s Algorithm that is optimized for a single destination. Dijkstra’s Algorithm can find paths to all locations; A\* finds paths to one location, or the closest of several locations. It prioritizes paths that seem to be leading closer to a goal.

### **Advantages:**

* A\* search algorithm is the best algorithm than other search algorithms.
* A\* search algorithm is optimal and complete.
* This algorithm can solve very complex problems.

### **Disadvantages:**

* It does not always produce the shortest path as it mostly based on heuristics and approximation.
* A\* search algorithm has some complexity issues.
* The main drawback of A\* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

7.Conclusion:- As a team we try our best to build a best path finding game called “House Attack”. In this report we mentioned the game’s aim, interface, functionality, user manual, tools and others things. we hope that this game will be enjoyable to its user.