

ANALYSIS AND DESIGN OF ALGORITHMS MINI PROJECT REPORT

**HEX MASTER**

**WITH**

**ARTIFICIAL INTELLIGENCE**

BY

**S J RAJATH KRISHNA**

**1PI13CS129**

**SAYAN GUHA**

**1PI13CS139**

**January 2015 – May 2015**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**PES INSTITUTE OF TECHNOLOGY**

**100 FEET RING ROAD, BANASHANKARI III STAGE**

**BANGALORE-560085**



**PES Institute of Technology,**

**100 Feet Ring Road, BSK 3rd Stage, Bangalore-560085**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**CERTIFICATE**

This is to certify that the project entitled **HEX MASTER WITH ARTIFICIAL INTELLIGENCE** is presented by us in partial fulfillment for the award of degree of **Bachelor of Engineering in Computer Science of the Visvesvaraya Technological University, Belgaum during the year 2014-2015**. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the report. The Mini Project has been approved as it satisfies the academic requirements of the course prescribed for the Bachelor of Engineering Degree.

Signature of the Guide Signature of the HOD

**(Mr. Channa Bankapur) (Prof. Nitin V Pujari)**

**ACKNOWLEDGEMENT**

It is a great pleasure for us to acknowledge the assistance and contribution of a large number of individuals from various quarters of this project.

We are grateful to our **Principal** **K S Sridhar** and HOD **Prof. Nitin V Pujari** for their support and providing all facilities for developing this project.

We acknowledge the unflinching support, guidance and motivation rendered by our lecturers, Mr.Channa Bankapur,Prof. Nagabushan Satya Kumar and Mrs.Savithri**,** as project guides of Department of Computer Science and Engineering. We express our profound gratitude for their continuous support and invaluable guidance in turning this project into reality. We are thankful for their interest and encouragement.

We also sincerely thank all the teaching and no teaching staff for their support and cooperation.

**TABLE OF CONTENTS**

1. **INTRODUCTION 5**
2. **IMPLEMENTATION DETAILS 6 - 7**
3. **RESULT 8**
4. **CONCLUSION & REFERENCES 9**

**INTRODUCTION**

|  |
| --- |
| Begin with a board, two players, and a supply of pieces or markers in two different colors. Each player chooses a color of markers and a pair of opposite sides which is shown here in red and blue.  **[Play Hex Step 1.jpg](http://www.wikihow.com/Play-Hex#/Image:Play-Hex-Step-1.jpg)**  Choose which player will go first |
| Take turns’ placing one piece on the board per move. Once a piece is played, it stays there for the rest of the game. Either player may play a piece in any hexagon that is not yet occupied  **[Play Hex Step 3.jpg](http://www.wikihow.com/Play-Hex#/Image:Play-Hex-Step-3.jpg)** |
| Win by connecting a row of adjacent hexagons between your two opposite sides. Any continuous path will do and the pieces need not have to be placed in order. Here red has won.  **[Play Hex Step 4.jpg](http://www.wikihow.com/Play-Hex#/Image:Play-Hex-Step-4.jpg)** |

**IMPLEMENTATION DETAILS**

**DATA STRUCTURES USED**

**The following data structures were used in developing this project.**

**1-D and 2-D Array**

**Collections**

**ArrayList**

**The ArrayList class extends AbstractList and implements the List interface. ArrayList supports dynamic arrays that can grow as needed. It is created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array may be shrunk.**

**We used ArrayList to populate the winning paths for the AI using the add function.**

**Queue**

**Queue is also an abstract data type or a linear data structure, in which the first element is inserted from one end called REAR (also called tail), and the deletion of existing element takes place from the other end called as FRONT (also called head). This makes queue as FIFO data structure, which means that element inserted first will also be removed first.**

**A queue is designed to have elements inserted at the end of the queue, and elements removed from the beginning of the queue. Just like a queue in a supermarket.**

**We used Queue to dequeue the row and column index for the next cell to be traversed in the path.**

**DFS**

**Depth-first search (DFS) is an** [**algorithm**](http://en.wikipedia.org/wiki/Algorithm) **for traversing or searching** [**tree**](http://en.wikipedia.org/wiki/Tree_data_structure) **or** [**graph**](http://en.wikipedia.org/wiki/Graph_%28data_structure%29) **data structures. One starts at the** [**root**](http://en.wikipedia.org/wiki/Tree_%28data_structure%29#Terminology) **(selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before** [**backtracking**](http://en.wikipedia.org/wiki/Backtracking)**.**

**It is like** [**preorder traversal of tree**](http://www.thecrazyprogrammer.com/2013/12/c-program-to-create-binary-tree-using-recursion.html)**. Traversal can start from any vertex, say Vi . Vi is visited and then all vertices adjacent to Vi are traversed recursively using DFS.**

**Since, a graph can have cycles. We must avoid revisiting a node. To do this, when we visit a vertex V, we mark it visited. A node that has already been marked as visited should not be selected for traversal. Marking of visited vertices can be done with the help of a global array visited[ ]. Array visited[ ] is initialized to false (0).**

**Algorithm for DFS**

n ← number of nodes

Initialize visited[ ] to false (0)

for(i=0;i<n;i++)

               visited[i] = 0;

void DFS(vertex i) [DFS starting from i]

{

               visited[i]=1;

               for each w adjacent to i

                              if(!visited[w])

                                             DFS(w);

}

**The DFS or Depth First Search algorithm is used to generate the winning paths and these paths are then populated into the arraylist which is then put into a queue and the row and column indices are popped out one after the other.**

**It is also used to determine if the path in the graph is connected to form a winning path.**

**Artificial Intelligence Implementation**

**The 11x11 board is converted to a graph with 121 vertices and a 121x121 adjacency matrix is formed from the connection between the surrounding hex cells. Depth First Search is then applied to this matrix to compute all the winning paths. This generates a game tree from which the appropriate node that leads to a winning leaf node is selected by a decision making algorithm and if the path is blocked out the algorithm backtracks to the appropriate next best node in the game tree.**

**RESULT**

Complexity of Depth First Search

O(|V2|) where V is the number of vertices

For 2 player case: Number of vertices ‘V’ is 11

For Artificial Intelligence case: Number of vertices ‘V’ is 121

Time taken to compute and generate all the paths (Only initial case) for the artificial intelligence case is: 12 seconds (approx), this varies for different cases

**CONCLUSION**

**This project has greatly helped us in analyzing various algorithms that would serve the purpose of finding connected components in a graph. We also learnt the basics of machine learning in the process of developing this project.**

**Scope for improvement**

**The AI developed is quite inefficient. We would definitely enhance the efficiency and also improve it further such that the computer’s play is on par with human intelligence.**

**REFERENCES**

[1]<http://www.codecademy.com/>

[2]<http://www.w3schools.com/js>

[3] http://www.codeproject.com

[4] https://www.vogella.com

[5] http://technophilia.co.in