Assignment 1 Advance Game Programming

All 5 Activities

Himank.Bansal

u3183058

# Activity 1

## 

Figure 1: Image Of Activity 1 working

## Pseudocode for A\* algorithm:

OpenNodes//Set Nodes to be checked

ClosedNodes// Set of nodes already visited

PossibleNodes//Neighbour nodes that might be added to the Open Nodes

StartNode// Start Position for search

EndNode// Final goal to be reached

Find Path():

* CurrentNode= StartNode;
* loop
  + Get PossibleNodes for (CurrentNode);// get all the walkable nodes around the current position
  + If the neighbour node is not walkable skip to next node
    - Get Left neighbour Node
    - Get Right neighbour Node
    - Get Up neighbour Node
    - Get Down neighbour Node
  + Set CostofTravel for (PossibleNodes) list
    - Set the Gcost for each Possible Node// distance from the start node
    - Set the Hcost for each Possible Node// distance from the goal according the type of tile
    - Set the Fcost for each Possible Node// total cost of travel
  + Set the parent of the node
    - If node parent!=null
      * If Node.parent.Gcost>currentNode.Gcost
        + Node.parent=CurrentNode
        + Node.Gcost.Update();
  + Check if the OpenNodes.contains(each Node)
    - If true continue;
    - Else OpenNode.Add(Node);
  + Remove the Current Node from Open Node list
  + Add the Current Node to the Closed Node list
* Select Next Node From Open Node list with the lowest Fcost
* Check if the Next Node is the Goal or EndNode
  + If yes trace the path
  + If no then Recursion re- call the FindPath()

# Activity 2

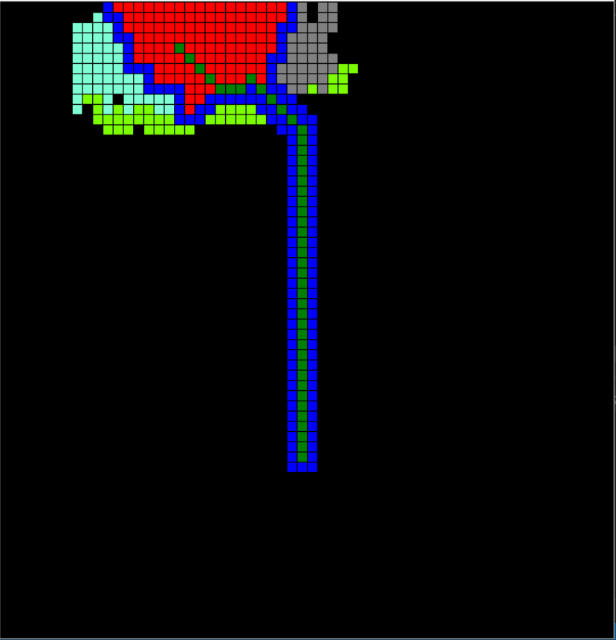


Figure 2: Image of Activity 2 A\* Diagonal Path Finding working

## Pseudocode for A\* algorithm:

OpenNodes//Set Nodes to be checked

ClosedNodes// Set of nodes already visited

PossibleNodes//Neighbour nodes that might be added to the Open Nodes

StartNode// Start Position for search

EndNode// Final goal to be reached

Find Path():

* CurrentNode= StartNode;
* loop
  + Get PossibleNodes for (CurrentNode);// get all the walkable nodes around the current position
  + If the neighbour node is not walkable skip to next node
    - Get Left neighbour Node
    - Get Right neighbour Node
    - Get Up neighbour Node
    - Get Down neighbour Node
    - Get TopLeft neighbour Node
    - Get Top Right Neighbour Node
    - Get Bottom Right Neighbour Node
    - Get Bottom Left Neighbour Node
  + Set CostofTravel for (PossibleNodes) list
    - Set the Gcost for each Possible Node// distance from the start node
    - Set the Hcost for each Possible Node// distance from the goal according the type of tile
    - Set the Fcost for each Possible Node// total cost of travel
  + Set the parent of the node
    - If node parent!=null
      * If Node.parent.Gcost>currentNode.Gcost
        + Node.parent=CurrentNode
        + Node.Gcost.Update();
  + Check if the OpenNodes.contains(each Node)
    - If true continue;
    - Else OpenNode.Add(Node);
  + Remove the Current Node from Open Node list
  + Add the Current Node to the Closed Node list
* Select Next Node From Open Node list with the lowest Fcost
* Check if the Next Node is the Goal or EndNode
  + If yes trace the path
  + If no then Recursion re- call the FindPath()

# Activity 3 Multi-Threading

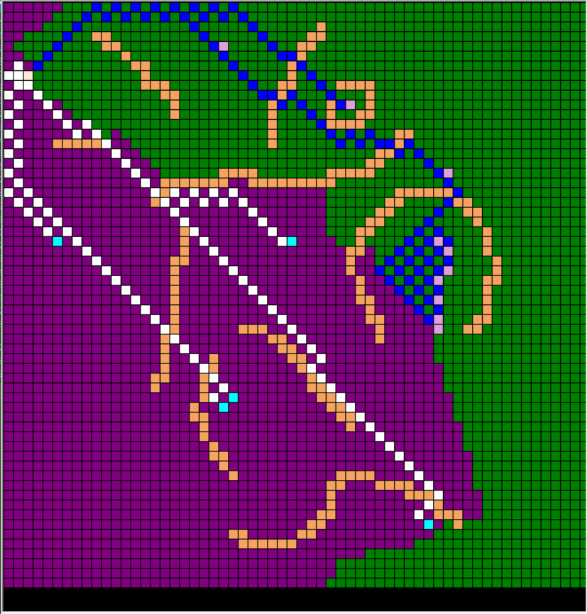


Figure 3:Image of Multi Agent Breadth First search in work

Green Agent 1 visited nodes

Purple agent 2 Visited Nodes

They Communicate the visited nodes to each other to save the processing

## Pseudo-Code for Multi agent search Using BFS(Breadth-First Search)

Start Node

Item

Agent 1 Thread

Agent 2 Thread

Closed Nodes// Concurrent Queue for visited nodes common between both the agents

Open Nodes// Concurrent Queue for Nodes to be visited

Set Agents ():

Start Two Threads:

Agent 1

Agent 2

Agent1FindItem(),Agent2FindItem():

Loop:

Set Current Node for Agent 1 and Agent 2 as the Start Node

Add Current Node to OpenNodes

Select the First Object in queue Open Nodes as Current Node

Get Possible Nodes for Agent 1 and Agent 2 using threads for CurrentNode for Agent 1 and Agent 2

Left

Right

Up

Down

Top Left

Top Right

Bottom Left

Bottom Right

Check if any possible node is an item

If yes

Check if item.Collected==true

If false Collect and mark the item node collected in the board;

If true continue;

If no

Continue;

Check foreach(Node in Possible Nodes)

If(OpenNodes.Contains(Node))

{

Continue;

}

Else

OpenNodes.Add(Node);

Add all the Possible Nodes to Open Nodes Queue

Dequeue Current Node for Agent 1 and Agent 2 from Open Nodes

Add the Current Node for Agent 1 and Agent 2 To Closed Nodes Queue

Check if(OpenNodes.Count>0)

If true

{

Do Recursion recall Agent1FindItem and Agent2FindItem according to Thread.CurrenthreadName  
}

If false

{

Stop the threads;

Stop the loop;

Send Message “Map Search Complete”

}

# Activity 4:Possible Path Detection on graph Of Shapes

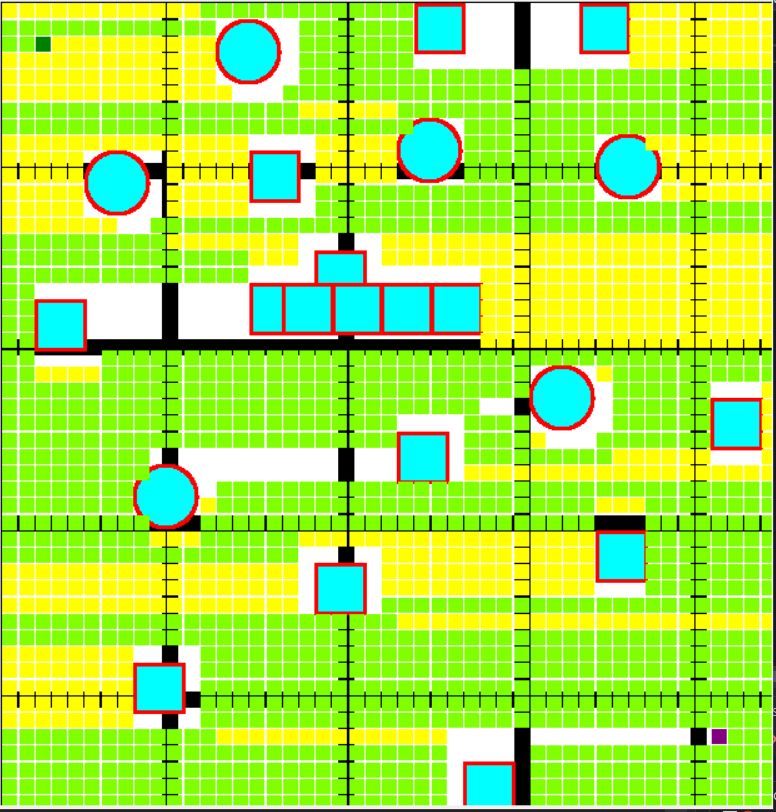


Figure 4: Image of Activity 4 working

## Pseudo-Code For Activity 4 Depth First Search

Current Node =StartNode

Open Node// Stack for Nodes top be visited

ClosedNode// Stack For Nodes that are visited and blocked

Push Current Node to OpenNode Stack

loop

FindGoal():

Get the closest possible node for Current Node

If(Possible Node!=null||PossibleNode is Obstacle)

If true:

Push Possible Node to Open Node Stack

Mark the Node Visited

Recall the same function and do recursion FindGoal();

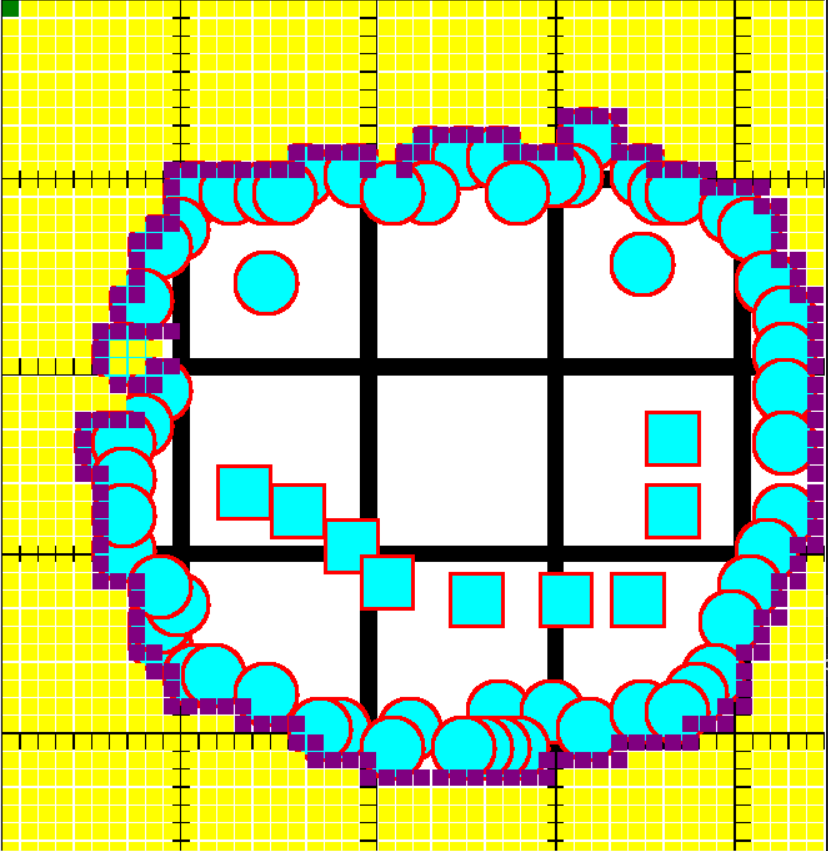
Else if(PossibleNode==null)

Pop the current node from the OpenNode Stack

If(OpenNode.count>0)

Recursion recall FindGoal();

# Activity 5: Shape Boundary Detection



## Pseudo Code for Shape Detection Using BFS Breadth First Search

Get Size of the main Grid

Divide each cell of main grid in 10\*10 grid:

CurrentNode=StartNode

StartDetection()

Current Node=OpenNode.First()

Get Possible Nodes for Current node

Check each node if coordinates if the node type is obstacle

If(node type obstacle)

Shape coordinate detected

Else:

Enqueue Node to Open Node

Enqueue Current Node to Closed nodes

CurrentNode=Dequeue Open Node

If(Open Node.Count>0)

Recursion Start Detection()

Used general equations of square and circle to get form a formula to detect if the Coordinate of sub Grid are Colliding with the Coordinate of the Shape in Main Grid with a n offset of difference between the size of both the grid;