CT255 [2D games in Java]
Sample Solution for A* Assignment

Main application class:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.image.*;
import java.io.*;

public class AStarMaze extends JFrame implements Runnable, MouseListener, MouseMotionListener,
KeyListener {

    // member data
    private boolean isInitialised = false;
    private BufferStrategy strategy;
    private Graphics offscreenBuffer;
    private boolean map[][] = new boolean[40][40];
    private boolean isGameRunning = false;
    private BadGuy badguy;
    private Player player;

    private String FilePath;

    // constructor
    public AStarMaze () {

        //Display the window, centred on the screen
        Dimension screensize = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        int x = screensize.width/2 - 400;
        int y = screensize.height/2 - 400;
        setBounds(x, y, 800, 800);
        setVisible(true);
        this.setTitle("A* Pathfinding Demo");

        FilePath = System.getProperty("user.dir") + "\\";
        System.out.println("Working Directory = " + FilePath);

        // load raster graphics and instantiate game objects
        ImageIcon icon = new ImageIcon(FilePath+"badguy.png");
        Image img = icon.getImage();
        badguy = new BadGuy(img);
        icon = new ImageIcon(FilePath+"player.png");
        img = icon.getImage();
        player = new Player(img);

        // create and start our animation thread
        Thread t = new Thread(this);
        t.start();

        // initialise double-buffering
        createBufferStrategy(2);
        strategy = getBufferStrategy();
        offscreenBuffer = strategy.getDrawGraphics();

        // register the Jframe itself to receive mouse and keyboard events
        addMouseListener(this);
        addMouseMotionListener(this);
        addKeyListener(this);

        // initialise the map state
        for (x=0;x<40;x++) {
            for (y=0;y<40;y++) {
                map[x][y]=false;
            }
        }

        isInitialised = true;
    }

    // thread's entry point
    public void run() {
        long loops=0;
        while ( 1==1 ) {
            // 1: sleep for 1/5 sec
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) { }
```

```
        // 2: animate game objects
        if (isGameRunning) {
            loops++;
            if (player.move(map)) // player (potentially) moves every frame
                badguy.reCalcPath(map,player.x,player.y);// player moved so badguy recalcs path
            if (loops%3==0) // badguy moves once every 3 frames
                badguy.move(map,player.x,player.y);
        }

        // 3: force an application repaint
        this.repaint();

    }
}

private void loadMaze() {
    String filename = FilePath+"maze.txt";
    String textinput = null;
    try {
        BufferedReader reader = new BufferedReader(new FileReader(filename));
        textinput = reader.readLine();
        reader.close();
    }
    catch (IOException e) { }

    if (textinput!=null) {
        for (int x=0;x<40;x++) {
            for (int y=0;y<40;y++) {
                map[x][y] = (textinput.charAt(x*40+y)=='1');
            }
        }
    }
}

private void saveMaze() {
    // pack maze into a string
    String outputtext="";
    for (int x=0;x<40;x++) {
        for (int y=0;y<40;y++) {
            if (map[x][y])
                outputtext+="1";
            else
                outputtext+="0";
        }
    }

   try {
      String filename = FilePath+"maze.txt";
      BufferedWriter writer = new BufferedWriter(new FileWriter(filename));
      writer.write(outputtext);
      writer.close();
   }
   catch (IOException e) { }
}

// mouse events which must be implemented for MouseListener
public void mousePressed(MouseEvent e) {
  if (!isGameRunning) {
     // was the click on the 'start button'?
     int x = e.getX();
     int y = e.getY();
     if (x>=15 && x<=85 && y>=40 && y<=70) {
         isGameRunning=true;
         badguy.reCalcPath(map,player.x,player.y); // initial path
         return;
     }
     // or the 'load' button?
     if (x>=315 && x<=385 && y>=40 && y<=70) {
         loadMaze();
            return;
     }
     // or the 'save' button?
     if (x>=415 && x<=485 && y>=40 && y<=70) {
         saveMaze();
         return;
     }
  }

  // determine which cell of the gameState array was clicked on
  int x = e.getX()/20;
  int y = e.getY()/20;
  // toggle the state of the cell
  map[x][y] = !map[x][y];
  if (isGameRunning)
      badguy.reCalcPath(map,player.x,player.y); // maze changed so badguy recalcs path
```

```
    // throw an extra repaint, to get immediate visual feedback
    this.repaint();
    // store mouse position so that each tiny drag doesn't toggle the cell
    // (see mouseDragged method below)
    prevx=x;
    prevy=y;
  }

  public void mouseReleased(MouseEvent e) {}

  public void mouseEntered(MouseEvent e) {}

  public void mouseExited(MouseEvent e) {}

  public void mouseClicked(MouseEvent e) {}
 //

  // mouse events which must be implemented for MouseMotionListener
  public void mouseMoved(MouseEvent e) {}

  // mouse position on previous mouseDragged event
  // must be member variables for lifetime reasons
  int prevx=-1, prevy=-1;
  public void mouseDragged(MouseEvent e) {
      // determine which cell of the gameState array was clicked on
      // and make sure it has changed since the last mouseDragged event
      int x = e.getX()/20;
      int y = e.getY()/20;
      if (x!=prevx || y!=prevy) {
          // toggle the state of the cell
          map[x][y] = !map[x][y];
          // throw an extra repaint, to get immediate visual feedback
          this.repaint();
          // store mouse position so that each tiny drag doesn't toggle the cell
          prevx=x;
          prevy=y;
          if (isGameRunning)
              badguy.reCalcPath(map,player.x,player.y); // maze changed so badguy recalcs path

      }
  }
  //

  // Keyboard events
  public void keyPressed(KeyEvent e) {
      if (e.getKeyCode()==KeyEvent.VK_LEFT)
          player.setXSpeed(-1);
      else if (e.getKeyCode()==KeyEvent.VK_RIGHT)
          player.setXSpeed(1);
      else if (e.getKeyCode()==KeyEvent.VK_UP)
          player.setYSpeed(-1);
      else if (e.getKeyCode()==KeyEvent.VK_DOWN)
          player.setYSpeed(1);
  }

  public void keyReleased(KeyEvent e) {
      if (e.getKeyCode()==KeyEvent.VK_LEFT || e.getKeyCode()==KeyEvent.VK_RIGHT)
          player.setXSpeed(0);
      else if (e.getKeyCode()==KeyEvent.VK_UP || e.getKeyCode()==KeyEvent.VK_DOWN)
          player.setYSpeed(0);
  }

  public void keyTyped(KeyEvent e) { }
  //

  // application's paint method
  public void paint(Graphics g) {
     if (!isInitialised)
         return;

     g = offscreenBuffer; // draw to offscreen buffer

     // clear the canvas with a big black rectangle
     g.setColor(Color.BLACK);
     g.fillRect(0, 0, 800, 800);

     // redraw the map
     g.setColor(Color.WHITE);
     for (int x=0;x<40;x++) {
         for (int y=0;y<40;y++) {
           if (map[x][y]) {
               g.fillRect(x*20, y*20, 20, 20);
            }
         }
     }
```

```
        // redraw the player and badguy
        // paint the game objects
        player.paint(g);
        badguy.paint(g);

        if (!isGameRunning) {
            // game is not running..
            // draw a 'start button' as a rectangle with text on top
            // also draw 'load' and 'save' buttons
            g.setColor(Color.GREEN);
            g.fillRect(15, 40, 70, 30);
            g.fillRect(315, 40, 70, 30);
            g.fillRect(415, 40, 70, 30);
            g.setFont(new Font("Times", Font.PLAIN, 24));
            g.setColor(Color.BLACK);
            g.drawString("Start", 22, 62);
            g.drawString("Load", 322, 62);
            g.drawString("Save", 422, 62);
        }

        // flip the buffers
        strategy.show();
    }

    // application entry point
    public static void main(String[] args) {
        AStarMaze w = new AStarMaze();
    }

}
```

Player class:

```
import java.awt.Graphics;
import java.awt.Image;

public class Player {

    Image myImage;
    int x=0,y=0;
    int xSpeed=0, ySpeed=0;

    public Player( Image i ) {
        myImage=i;
        x=10;
        y=35;
    }

    public void setXSpeed( int x ) {
        xSpeed=x;
    }

    public void setYSpeed( int y ) {
        ySpeed=y;
    }

    public boolean move(boolean map[][]) {
        int newx=x+xSpeed;
        int newy=y+ySpeed;
        if ((xSpeed!=0 || ySpeed!=0) && !map[newx][newy]) {
            x=newx;
            y=newy;
            return true;
        }
        return false;
    }

    public void paint(Graphics g) {
        g.drawImage(myImage, x*20, y*20, null);
    }

}
```

Node class:

```
//A* node

public class node {
    // open/closed states
    public boolean isOpen;
    public boolean isClosed;
    // position of parent
    public int parentx,parenty;
    // A* data
```

```
    public int f,g,h;
    // position on map
    public int x,y;
}
```

BadGuy class:

```
import java.awt.Graphics;
import java.awt.Image;
import java.io.Console;
import java.util.*;
public class BadGuy {

    Image myImage;
    int my_x=0, my_y=0;
    boolean hasPath=false;
    node nodes[][] = new node[40][40]; // 2D array of A* nodes
    Stack path = null; // stack of A* nodes
    LinkedList openList = null; // linked-list of A* nodes

    public BadGuy( Image i ) {
        myImage=i;
        my_x = 30;
        my_y = 10;
        path = new Stack();
        openList = new LinkedList();
        // initialise array of nodes
        for (int x=0;x<40;x++) {
            for (int y=0;y<40;y++) {
                nodes[x][y] = new node();
                nodes[x][y].x = x;
                nodes[x][y].y = y;
            }
        }
    }

    public int manhattanDist(int x1,int y1,int x2, int y2) {
        return Math.abs(x1-x2)+Math.abs(y1-y2);
    }

    public void reCalcPath(boolean map[][],int targx, int targy) {
        // calculate A* path to targx,targy, taking account of walls defined in map[][]
        System.out.print("Pathfinding from "+my_x+","+my_y+" to "+targx+","+targy+"\n");

        // reset array of nodes
        for (int x=0;x<40;x++) {
            for (int y=0;y<40;y++) {
                // mark node as closed if a wall is at x,y
                nodes[x][y].isOpen = false;
                nodes[x][y].isClosed = map[x][y];
            }
        }

        // reset linked-list and stack
        openList.clear();
        path.clear();

        // initial step: mark badguy's current position as open
        nodes[my_x][my_y].isOpen = true;
        nodes[my_x][my_y].g = 0; // cost from start to here
        nodes[my_x][my_y].h = manhattanDist(my_x,my_y,targx,targy); // estimated cost from here to
target
        nodes[my_x][my_y].f = nodes[my_x][my_y].g+nodes[my_x][my_y].h; // f=g+h
        openList.add(nodes[my_x][my_y]); // add node to open list

        boolean finished = false, givenUp = false;

        while (!finished && !givenUp) {
            // general steps:
            // step 1: find most promising open node (with lowest f value)
            node best_node = null;
            Iterator i = openList.iterator();
            while (i.hasNext()) {
                node n = (node)i.next();
                if (best_node==null || best_node.f>n.f)
```

```
                    best_node = n;
            }

            // step 2: expand most promising node by opening its neighbours
            if (best_node!=null) {
                testneighbours: // this is a line label we can break to
                for (int xx=-1;xx<=1;xx++) {
                    for (int yy=-1;yy<=1;yy++) {
                        int xxx=best_node.x+xx, yyy=best_node.y+yy;
                        if (xxx>=0 && xxx<40 && yyy>=0 && yyy<40 && (xx==0 || yy==0)) { // avoid out-
of-bounds!, also disallow diagonals
                            if (!nodes[xxx][yyy].isClosed && !nodes[xxx][yyy].isOpen) {
                                // open node and set its parent to the node that's currently being
expanded
                                nodes[xxx][yyy].isOpen=true;
                                openList.add(nodes[xxx][yyy]);
                                nodes[xxx][yyy].parentx = best_node.x;
                                nodes[xxx][yyy].parenty = best_node.y;
                                // cost from start to here, i.e. 1 greater than cost from start to
parent
                                nodes[xxx][yyy].g = 1 + best_node.g;
                                // estimated cost from here to target
                                nodes[xxx][yyy].h = manhattanDist(xxx,yyy,targx,targy);
                                nodes[xxx][yyy].f = nodes[xxx][yyy].g + nodes[xxx][yyy].h;
                                // special case: have we found our target?
                                if (xxx==targx && yyy==targy) {
                                    finished=true;
                                    break testneighbours; // break out of both nested loops
                                }
                            }
                        }
                    }
                }
                // close the node we have just expanded
                best_node.isOpen = false;
                best_node.isClosed = true;
                openList.remove(best_node);
            }
            else {
                // openList was empty => maze is unsolvable
                givenUp = true;
                System.out.print("No path found!\n");
            }
        }

        if (finished) {
            // now construct our path as a stack (LIFO => easy to reverse)
            System.out.print("Now at "+my_x+","+my_y+" ..push path: ");
            int x = targx, y = targy;
            while (x!=my_x || y!=my_y) {
                path.push(nodes[x][y]);
                int parentx = nodes[x][y].parentx;
                int parenty = nodes[x][y].parenty;
                System.out.print(x+","+y+" ");
                x = parentx;
                y = parenty;
            }
            System.out.print("\n");
            hasPath = true;
        }
        else
            hasPath = false;
    }

    public void move(boolean map[][],int targx, int targy) {
        if (hasPath) {
            // follow A* path, if we have one defined
            node nextNode = (node)path.pop();
            targx = nextNode.x;
            targy = nextNode.y;
            if (path.size()==0)
                hasPath=false;
            System.out.print("Popped "+targx+","+targy+"\n");
            my_x = targx;
```

```
            my_y = targy;
        }
        else {
            // no path known, so just do a dumb 'run towards' behaviour
            int newx = my_x, newy = my_y;
            if (targx<my_x)
                newx--;
            else if (targx>my_x)
                newx++;
            else if (targy<my_y)
                newy--;
            else if (targy>my_y)
                newy++;
            if (!map[newx][newy]) { // blocked by walls
                my_x = newx;
                my_y = newy;
            }
        }
    }

    public void paint(Graphics g) {
        g.drawImage(myImage, my_x*20, my_y*20, null);
    }

}
```