IT- 326 Artificial Intelligence

Final Document

Angry Birds – Finishers

Pavan Thatha (201101024), AR SAI ANIRUDH (201201197), M CHARITH REDDY (201201200)

Flow & Thought Process of overall strategy:

A) As we played the game multiple times we observed that there were several points in the structure, hitting on which was causing maximum and immediate destruction with the death of the pig. Even in a very hefty and populated structure such points were useful in getting down the structure. We are naming such points as "**weak points**" in the structure.
The logic behind finding the weak points in a structure is that if we have a horizontal long block supported by some unstable long vertical blocks then the intersection point of these blocks can be considered as weak points because the lead to a considerable amount of tremor or destruction in the structure.

Fig 1: Weak Point

But there is a small anomaly in this hitting the weak point's strategy. Suppose we keep on hitting the weak points but if a weak point is far away from a pig then there is no significance in choosing that weak point. For this purpose we divided the whole structure into sub-structures and targeted only that particular substructure known as Admissible Substructure. This is how we came up with the idea of "**sub-structures**".
For finding the sub-structures we are using the isConnected method which returns a Boolean true when two blocks/objects are in contact.

Further refining the list of sub-structures. The **admissible substructures** are defined as those sub-structures which consist of pigs and which are closer to the sling where the bird is being released.

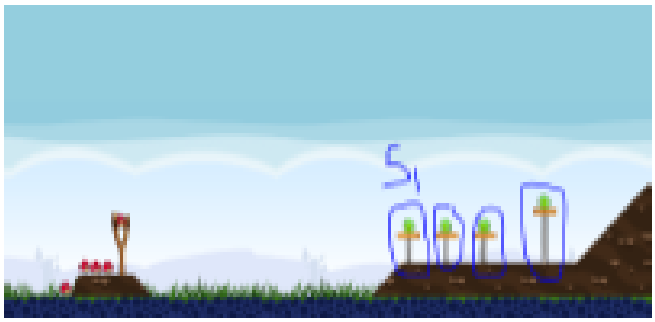

Fig 2: Sub-structures

Codes Related to A:

Weak Points Method:

// Returns weak points of the above form, present in a structure.

```
public ArrayList<Point> WeakPoints_Vertical_blocks_on_Horizontal(List<ABObject> blocks) {
    ArrayList<Point> weak_points = new ArrayList<Point>();
    ArrayList<Point> weak_points1 = new ArrayList<Point>();
    // ArrayList<ABObject> weak_blocks=new ArrayList<ABObject>();
    for (int i = 0; i < blocks.size(); i++) {
       int nblocks=0;
       ABObject block = blocks.get(i);
       Dimension size = block.getSize();
       Point center = block.getCenter();
       int wi = (int) size.width;
       int hi = (int) size.height;
       Point top = new Point(center.x - (wi / 2), center.y - (hi / 2));
```

```
        Point bottom = new Point(center.x - (wi / 2), center.y + (hi / 2));
        if (wi>hi/*||Math.abs(top.y-bottom.y)<10*/) {
           //System.out.println("iffff");
           for (int j = 0; j < blocks.size(); j++) {
              ABObject block1 = blocks.get(j);
              Dimension size1 = block1.getSize();
              Point center1 = block1.getCenter();
              int wi1 = (int) size1.width;
              int hi1= (int) size1.height;
              Point top1 = new Point(center1.x - (wi1 / 2), center1.y - (hi1 / 2));
              Point bottom1 = new Point(center1.x - (wi1 / 2), center1.y + (hi1 / 2));
              //System.out.println("Height"+hi1+"width"+wi1);;
              if(block1!=block){
                 if(top1.y>bottom.y &&
hi1>wi1/*&&&&(bottom1.x>=top.x)&&(bottom1.x<=bottom.x)*/&&isconnected(block,block1))
                 {


                    nblocks++;
                    Point pt=center;
                    pt.x=top.x;
                    pt.y=top.y+hi;
                    weak_points1.add(pt);
                 }
              }
           }
           if(nblocks==1||nblocks==2){
              Point pt=center;
              pt.x=top.x;
              pt.y=top.y+hi;
```

```
                weak_points.add(pt);
            }


        }
    }
    if(weak_points.size()==0){
        weak_points=weak_points1;
    }
   /* if(weak_points.size()==0){
        //center of mass
    }*/
    return weak_points;
}
```

IsConnected Method :

// Whether two objects are in contact with each other.

//Done using substituting points of one block in a four straight line equations of another block.

```
public  boolean isconnected(ABObject o, ABObject o1) {


    Dimension size1 = o.getSize();
    Point center1 = o.getCenter();
    int wi1 = (int) size1.width;
    int hi1 = (int) size1.height;
    Point ltop1 = new Point(center1.x - (wi1 / 2), center1.y - (hi1 / 2));
    Point lbottom1 = new Point(center1.x - (wi1 / 2), center1.y + (hi1 / 2));
    Point rtop1 = new Point(center1.x + (wi1 / 2), center1.y - (hi1 / 2));
```

```
Point rbottom1 = new Point(center1.x + (wi1 / 2), center1.y + (hi1 / 2));
Dimension size2 = o1.getSize();
Point center2 = o1.getCenter();
int wi2 = (int) size2.width;
int hi2 = (int) size2.height;
Point ltop2 = new Point(center2.x - (wi2 / 2), center2.y - (hi2 / 2));
Point lbottom2 = new Point(center2.x - (wi2 / 2), center2.y + (hi2 / 2));
Point rtop2 = new Point(center2.x + (wi2 / 2), center2.y - (hi2 / 2));
Point rbottom2 = new Point(center2.x + (wi2 / 2), center2.y + (hi2 / 2));



Point Pts2[] = new Point[4];
Pts2[0] = ltop2;
Pts2[1] = lbottom2;
Pts2[2] = rtop2;
Pts2[3] = rbottom2;
Point Pts1[] = new Point[4];
Pts1[0] = ltop1;
Pts1[1] = lbottom1;
Pts1[2] = rtop1;
Pts1[3] = rbottom1;
/*System.out.println("Object1 left top:"+ltop1+"Objec1 right bottom:"+rbottom1);
System.out.println("Object2 left top:"+ltop2+"Objec2 right bottom:"+rbottom2);*/
int lt = 0, lb = 1, rt = 2, rb = 3;
int xx, yy;
for (int i = 0; i < 4; i++)
{
   xx = Pts2[i].x;
   yy = Pts2[i].y;
```

```
        for (int j = 0; j < 4; j++)
        {
          if ((ltop1.x - rtop1.x) * (yy - rtop1.y) == (ltop1.y - rtop1.y) * (xx - rtop1.x) ||
              (ltop1.x - lbottom1.x) * (yy - lbottom1.y) == (ltop1.y - lbottom1.y) * (xx -
lbottom1.x) ||
              (rbottom1.x - lbottom1.x) * (yy - lbottom1.y) == (rbottom1.y - lbottom1.y) * (xx -
lbottom1.x) ||
              (rbottom1.x - rtop1.x) * (yy - rtop1.y) == (rbottom1.y - rtop1.y) * (xx - rtop1.x))
          {

            return true;
          }
        }
      }
      return false;
    }
```

Sub-Structures Method :

// Uses isConnected Method to divide whole structure into substurctures.

Used Dynamic programing by adding objects one by one to the Substructures list and then found all the connections (con) indexes of the list and merge those which are connected to the object and add back this SubStructure to the list and we finally get all the SubStructures List.

```
public void createSubStructures(){
      for(int i=0;i<objects.size();i++){
        //System.out.println("C1");
```

```
ABObject o=objects.get(i);
if(i==0){
   SubStructure ss=new SubStructure();
   ss.add(o);
   list.add(ss);
   continue;
}
ArrayList<Integer> con=new ArrayList<Integer>();
for(int j=0;j<list.size();j++){
   SubStructure ss1=list.get(j);
   for(int k=0;k<ss1.obj.size();k++){
      //System.out.println("C5");
      ABObject o1=ss1.obj.get(k);

      if(this.isconnected(o,o1)){
         con.add(j);
         break;
      }
   }
}
if(con.size()==0){
   SubStructure ss2=new SubStructure();
   ss2.add(o);
   list.add(ss2);
   continue;
}
//SubStructure temp=list.get(con.get(0));
int tem=con.get(0);
list.get(tem).add(o);
```

```
        for(int j=1;j<con.size();j++){
            SubStructure temp1=list.get(con.get(j));
            for(int k=0;k<temp1.obj.size();k++){
                //System.out.println("C2");
                list.get(con.get(0)).add(temp1.obj.get(k));
            }
        }
        for(int j=1;j<con.size();j++){
            //System.out.println("C3");
            int x=con.get(j);
            //System.out.println("C55");
            if(x>=list.size()){
                continue;
            }
            list.remove(x);
            //System.out.println("C55e");
        }
    }
}
```

Admissible Sub-Structures method:

// Returns Left most and max blocks substructure.

```
public void createBorder(){
    left=100000;right=-10;top=10000000;bottom=-10;
    for(int i=0;i<obj.size();i++){
        ABObject o=obj.get(i);
        Dimension size1 = o.getSize();
        Point center1 = o.getCenter();
```

```
        int wi1 = (int) size1.width;
        int hi1 = (int) size1.height;
        Point ltop1 = new Point(center1.x - (wi1 / 2), center1.y - (hi1 / 2));
        Point lbottom1 = new Point(center1.x - (wi1 / 2), center1.y + (hi1 / 2));
        Point rtop1 = new Point(center1.x + (wi1 / 2), center1.y - (hi1 / 2));
        Point rbottom1 = new Point(center1.x + (wi1 / 2), center1.y + (hi1 / 2));
        if(ltop1.x<left){
           left=ltop1.x;
        }
        if(ltop1.y<top){
           top=ltop1.y;
        }
        if(rbottom1.x>right){
           right=rbottom1.x;
        }
        if(rbottom1.y>bottom){
           bottom=rbottom1.y;
        }
      }
   }
```

SubStructure having Pig method :

// If a particular SubStructure has a pig In It.

//Done by creating border to each sub structure and checking for pig in it.

```
public boolean hasPigInIt(SubStructure ss){
      for(int i=0;i<pigs.size();i++){
         Point pig=pigs.get(i).getCenter();
         if(pig.x>=ss.left && pig.x<=ss.right && pig.y>=ss.top && pig.y <= ss.bottom){
```

```
            return true;
        }
    }
    return false;
}
```

B) Second alternative to pass the level is to hit pig wise. To know the chances of a pig to be smashed we are calculating an angle known as compulsory hit angle. We refined the naïve agent using this angle. To calculate this angle we have initially given birds a fixed potential according to its characteristic strengths and powers to hit a particular obstacle.

1. We consider the different angles pointing the pig and for each angle we run the "obstacle Detector" method which returns all the blocks present in the way of the bird to the pig.
2. After we get the obstacle blocks we determine the type of the block and reduce the potential of the bird accordingly. Potential required by each bird to hit a particular type of block depend on three parameters
    - Type of block
    - Type of Bird
    - Block Thickness

Blocks	Red	Blue	Yellow	Blue
Glass	650	750	200	500
Wood	800	750	250	500
Stone	1050	1000	1000	500

TABLE I. POTENTIALS REQUIRED BY BIRDS TO BREAK THE OBJECT

Note: Thickness factor is not mentioned in the above table. It has been directly implemented in the code.

3. For example, let's consider a yellow bird and three wooden blocks as obstacles in the path to the pig. We shoot the yellow bird towards these blocks with a potential of 1000 initially, which reduces by 250 for every wooden block, after which it has still a potential of 250 remaining (1000 – 3 * 250). Since the bird has potential left it has a high probability of hitting/smashing the pig. Hence, the angle of projection for this particular trajectory can be considered as a compulsory hit angle.

Codes Relevant to B

Obstacle Detector Method:

//Returns all the objects present in the path given a pig and an angle.

//Iterating through each block and checking if it exists in trajectory points path.

```
public ArrayList<ABObject> obstacleDetector(ArrayList<ABObject> objs,Rectangle sling,Double ang, ABObject pig)

  {

     TrajectoryPlanner tp=new TrajectoryPlanner();


     ArrayList<Point> edgesl = new ArrayList<Point>();

     ArrayList<Point> edgesr = new ArrayList<Point>();

     ArrayList<ABObject> blcks = new ArrayList<ABObject>();

     Point releasePoint = tp.findReleasePoint(sling, ang);//estimates the release point according to the given angle


     List<Point> traj1 = new ArrayList<Point>();

     // List<Point> traj2 = new ArrayList<Point>();

     traj1 = tp.predictTrajectory(sling, releasePoint);// predicts the trajectory according to the slingshot and the release point

     for(int i =0 ;i < objs.size();++i) // this is the loop for getting the top left and top right corner of a blck they are being stored in.
```

```
    {
        Dimension size1 = objs.get(i).getSize();
        Point center1 = objs.get(i).getCenter();
        int wi1 = (int) size1.width;
        int hi1 = (int) size1.height;
        Point ltop1 = new Point(center1.x - (wi1 / 2), center1.y - (hi1 / 2));
        Point lbottom1 = new Point(center1.x - (wi1 / 2), center1.y + (hi1 / 2));
        Point rtop1 = new Point(center1.x + (wi1 / 2), center1.y - (hi1 / 2));
        Point rbottom1 = new Point(center1.x + (wi1 / 2), center1.y + (hi1 / 2));
        int minx = Math.min(Math.min(ltop1.x, rtop1.x), Math.min(lbottom1.x, rbottom1.x));
        int maxx = Math.max(Math.max(ltop1.x, rtop1.x), Math.max(lbottom1.x, rbottom1.x));
        int miny = Math.min(Math.min(ltop1.y, rtop1.y), Math.min(lbottom1.y, rbottom1.y));
        int maxy = Math.max(Math.max(ltop1.y, rtop1.y), Math.max(lbottom1.y, rbottom1.y));
        {
            for (int j = 0; j < traj1.size(); ++j) {
                if(traj1.get(i).x < pig.getCenter().x)
                    if ((minx <= traj1.get(i).x && maxx>= traj1.get(i).x) && (miny <= traj1.get(i).y && maxy >=
traj1.get(i).y) ) {
                        blcks.add(objs.get(j));
                        break;
                    }
            }
        }
    }

    return blcks;
  }
```

Compulsary Hit Method:

//Returns true if probability of smashing a pig in a particular path is more.

// assigning potentials to each combination of bird and obstacle and subtracting during each obstacle in the path and determining compulsory hit on the basis on remaining potential.

```
public boolean iscompalsoryHit(ArrayList<ABObject> objs,ABObject pig,double angle,Rectangle
sling,ABObject bird) {
    ArrayList<ABObject> pigs=new ArrayList<ABObject>();
    pigs.add(pig);
    Finisher fin = new Finisher(objs,pigs, sling);
      TrajectoryPlanner tp = new TrajectoryPlanner();
      ArrayList<Point> pts = tp.estimateLaunchPoint(sling, pig.getCenter());
      double releaseAngle = tp.getReleaseAngle(sling, pig.getCenter());
      ArrayList<ABObject> blks = fin.obstacleDetector(objs, sling, releaseAngle, pig);
      if (sling != null) {
        int pot=1000;
        int y=0;
        if(bird.type.equals("blue")){
            y=1;
        }
        if(bird.type.equals("black")){
            y=3;
        }
        if(bird.type.equals("yellow")){
            y=2;
        }
        if(bird.type.equals("red")){
            y=0;
        }
        for (int j=0;j<blks.size();j++) {
          if(pot<=0){
```

```
                return false;
            }
            if(blks.get(j).type.equals("wood")) {
               pot = pot - potential[wood][y];
            }
            if(blks.get(j).type.equals("glass")) {
               pot = pot - potential[glass][y];
            }
            if(blks.get(j).type.equals("stone")) {
               pot = pot - potential[stone][y];
            }

        }
      }
   return true;
}
```

C) When we are unable to find the weak points and the compulsory hit pigs the next alternative is to hit the "center of mass" of the structure. The center of mass of a structure or a sub-structure is calculated in the method centerOfMass. By assigning various weight to the different block types and the positions of the blocks which are already known we can calculate the center of mass using the formulae in general physics.

Code Relevant to C

Center of mass method:

// Returns center of mass of whole structure by assigning weights to individual block types.

```
public Point centreOfMass(List<ABObject> blocks){
```

```
    int ice_wt=25,wood_wt=30,stone_wt=50;

    int cmx=0,cmy=0;
    int numx=0,dinx=0;
    int numy=0,diny=0;
    for(int i=0;i<blocks.size();i++){
       ABObject block=blocks.get(i);
       Dimension size=block.getSize();
       Point center=block.getCenter();
       if(block.getType().id==10){
          numx=numx+center.x*size.width*size.height*ice_wt;
          dinx+=size.width*size.height*ice_wt;
          numy=numy+center.y*size.width*size.height*ice_wt;
          diny+=size.width*size.height*ice_wt;
          continue;
       }
       if(block.getType().id==11){
          numx=numx+center.x*size.width*size.height*wood_wt;
          dinx+=size.width*size.height*wood_wt;
          numy=numy+center.y*size.width*size.height*wood_wt;
          diny+=size.width*size.height*wood_wt;
          continue;
       }
       if(block.getType().id==12){
          numx=numx+center.x*size.width*size.height*stone_wt;
          dinx+=size.width*size.height*stone_wt;
          numy=numy+center.y*size.width*size.height*stone_wt;
          diny+=size.width*size.height*stone_wt;
          continue;
       }
    }
    //System.out.println("x"+" "+numx/dinx +"  "+"y"+numy/diny);
    if(dinx==0 || diny==0){
       return new Point(0,0);
    }
    return new Point(numx/dinx,numy/diny);
}
```

D) Score Board:

Level 1 : 30930
Level 2 :52420
Level 3 :41270

Level 4 :30640
Level 5 :58390
Level 6 :36630
Level 7 :45680
Level 8 :49490
Level 9 :50660
Level 10 :51480
Level 11 :56150
Level 12 :57890
Level 13 :49170
Level 14 :64200
Level 15 :46400
Level 16 :51850
Level 17 :47960
Level 18 :53850
Level 19 :38930
Level 20 :40420
Level 21 :64070

Final Score : 10,18,480

E) Improvements that can be done:

We can update the values of the breaking potential table (Characteristic table) of the block with-respective bird strength using Reinforcement Learning.