```csharp
using Final_Project;
using Final_Project.Properties;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Timers;
using static Final_Project.Form1;

namespace BasicAI
{
    public class Enemy
    {
        //amalgamates most things required to make an enemy so most things are self contained,
        //same thing as gunner but doesnt shoot and just tries to follow player
        public PictureBox picEnemy;
        PictureBox player;
        Label[] projectiles;
        Boolean isActive = true;
        int internalHealth;
        public int healthcap;
        public int internalDeaths = 0;
        public double speed;
        public WeaponSelected currentWeapon;

        Form1 form = Form1.getInstance();

        System.Random r = new System.Random((int)System.DateTime.Now.Ticks);

        public Enemy(PictureBox picEnemy, PictureBox player, Label[] projectiles, double speed, int healthcap)
        {
            this.picEnemy = picEnemy;
            this.player = player;
            this.projectiles = projectiles;

            internalHealth = healthcap;
            this.healthcap = healthcap;
            this.speed = speed;
            currentWeapon = WeaponSelected.SWORD;
        }

        public void Update()
        {
            if (internalHealth > 0)
            {
                pursuit();
            }

            else
            {
                internalDeaths++;
                respawn(r.Next(0, 1000), r.Next(0, 1000));

            }
        }

        private void pursuit()
        {
            double posX = picEnemy.Left + (speed * getDeltaX());
            double posY = picEnemy.Top + (speed * getDeltaY());

            if (enemyTouch() || swordTouch())
            {
                posX -= (1 * getDeltaX());
                posY -= (1 * getDeltaY());

                picEnemy.Left = (int)posX;
                picEnemy.Top = (int)posY;
            }



            if (!shieldTouch())
            {
                picEnemy.Left = (int)posX;
                picEnemy.Top = (int)posY;
            }
            else
            {
                posX -= (0.1 * getDeltaX());
                posY -= (0.1 * getDeltaY());

                picEnemy.Left = (int)posX;
                picEnemy.Top = (int)posY;
            }

        }

        public void reset()
        {
            internalDeaths = 0;
            internalHealth = healthcap;
        }

        public void move(int x, int y)
        {
            picEnemy.Left = x;
            picEnemy.Top = y;
        }

        public void respawn(int x, int y)
        {
            internalHealth = healthcap;
            move(x, y);
        }

        public Boolean shieldTouch()
        {
            if (currentWeapon == WeaponSelected.SHIELD)
            {
                for (int i = 0; i < projectiles.Length; i++)
                {
                    if (picEnemy.Bounds.IntersectsWith(projectiles[i].Bounds))
                    {
                        return true;
                    }

                }
                return false;
            }
            else
                return false;
        }

        public Boolean enemyTouch()
        {
            return picEnemy.Bounds.IntersectsWith(player.Bounds);
        }

        public Boolean swordTouch()
        {

            for(int i = 0; i < projectiles.Length; i++)
            {
                if (currentWeapon == WeaponSelected.SWORD && picEnemy.Bounds.IntersectsWith(projectiles[i].Bounds))
                {
                    //form.spawnHeart();
                    internalHealth--;
                    return true;
                }
            }

            return false;
        }

        public int getHealth()
        {
            return internalHealth;
        }

        public int getDeaths()
        {
            return internalDeaths;
        }

        private double getDeltaY()
        {
            int playerX = player.Left;// - (player.Width / 2);
            int playerY = player.Top;// + (player.Height / 2);

            int enemyX = picEnemy.Left;// - (picEnemy.Width / 2);
            int enemyY = picEnemy.Top;// + (picEnemy.Height / 2);

            double deltaX = playerX - enemyX;
            double deltaY = playerY - enemyY;

            return deltaY;

        }

        private double getDeltaX()
        {
            int playerX = player.Left;// - (player.Width / 2);
            int playerY = player.Top;// + (player.Height / 2);

            int enemyX = picEnemy.Left;// - (picEnemy.Width / 2);
            int enemyY = picEnemy.Top;// + (picEnemy.Height / 2);

            double deltaX = playerX - enemyX;
            double deltaY = playerY - enemyY;

            return deltaX;

        }
    }
}

```