Henry Dunphy
6/12/2021

Babaroga Test

1) Warm-Ups

How many bits are there in a 32-bit integer?

Answer: There are 32 bits in a 32-bit integer.

How many bytes are there in 1 gigabyte?

1000 Megabytes in a gigabyte
1000 Kilobytes in a Megabyte -> 1,000,000 Kilobytes in a gigabyte
1000 Bytes in a kilobyte -> 1,000,000 Bytes in a MegaByte
Answer: 1,000,000,000 or 10^9 Bytes in a gigabyte

If a woodchuck chucks 2 wood a day, but improves his chucking ability by 50% every 2 days, how many days would it take him to chuck 1000 woods? This is assuming a woodchuck can in fact chuck wood.

D(0) = 2 wood
D(1) = 2 wood
D(2) = D(1) + (D(1) * .5) = 3 wood
D(3) = D(2) = 3 wood
D(4) = 4.5 wood (assuming you can throw half a wood)

Exponential growth formula: $x(t) = x_0 * (1 + r)^t$ (this is when t = 2 days)

$x(t) = 2 * (1.5)^t$
$1000 = 2 * (1.5)^t$
$500 = 1.5^t$
$\log_{1.5}(500) = \log_{1.5}(1.5^t)$
$\ln(500)/\ln(1.5) = t * 1$
$t = 15.327$ or after the 30th day

Answer: if a woodchuck could chuck wood, he could chuck over 1000 woods a day on the 31st day.

2) Object-Oriented Design

Chicken: IAnimal
-float hunger -int age -bool isMale -bool hasEggs -Roost home
-float Eat(Feed) -void Drink() -Egg[] LayEggs() -void Mate(Chicken) -void Die()

In my farm simulation the chicken class would inherit from an interface IAnimal which all the farm animals would derive from. This could give them common fields like hunger, age, and gender as well as some common functionality like Eat, Drink, Mate, and Die. This would allow the other systems to integrate seamlessly with all the farm animals making it easier to code, but also in game making things standardized.

In order to keep animals healthy they would need to eat and drink water each would be its own class that would inherit from another interface consumable. One of the ways the player would progress would be to improve the farm conditions which would start with better quality food and water. Better quality food/water would keep your farm animals alive for longer, but would cost more. So the IConsumable interface would have a quality and cost field to help accomplish this interaction.

The food for each animal would be grown as a crop which would require its own in-depth system, but I will focus on the animals for now. Everything from seeds, farm equipment, new habitats, animals, and water can be bought from the FarmersMarket. The Farmers Market would be where the player goes to buy new goods needed to expand and grow their farm as well as to sell off any of the products created on the farm such as eggs, crops and meat. Each associated MarketItem can be bought and sold in the FarmersMarket and will have an assigned value for cost. For simplicity's sake each MarketItem's cost is kept in a lookup table and will be constant.

The last system will be spaces. An interface ISpace can be either an animal habitat, crop field, or other area to be added later. IHabitat will inherit from ISpace and that will have common fields like list of IAnimal contents, spaces available, amount of food, amount of water, and compatible animal types. For example a Roost would be an IHabitat built for Chickens and depending on the price of the Roost you can hold more chickens, store more food and water and overall be more efficient.

3) 2D Math:

See attached file: Program.cs

Pasted at the end of the doc highlighting the required portions

4) Image Cropper

Executable in BmpCropperRunnable.zip

Visual Studio Code in BmpCropperCode.zip

```
struct Rectangle
{
    Point p1, p2, p3, p4;

    public Rectangle(Point pa, Point pb, Point pc, Point pd)
    {
        p1 = pa; p2 = pb; p3 = pc; p4 = pd;
    }

    public void Rotate(Point origin, float angle)
    {
        double angleRadians = angle * (Math.PI / 180);
        float sinTheta = (float)Math.Round(Math.Sin(angleRadians), 15);
        float cosTheta = (float) Math.Round(Math.Cos(angleRadians), 15);

        p1.RotatePoint(origin, cosTheta, sinTheta);
        p2.RotatePoint(origin, cosTheta, sinTheta);
        p3.RotatePoint(origin, cosTheta, sinTheta);
        p4.RotatePoint(origin, cosTheta, sinTheta);
    }

    public void Print()
    {
        Console.WriteLine($"{p1} {p2} {p3} {p4}");
    }
}

struct Point
{
    float x, y;
    public Point(float _x, float _y)
    {
        x = _x; y = _y;
    }

    public void RotatePoint(Point origin, float cosTheta, float sinTheta)
    {
        float _x = x - origin.x;
        float _y = y - origin.y;

        x = (_x * cosTheta - _y * sinTheta) + origin.x;
        y = (_x * sinTheta + _y * cosTheta) + origin.y;
    }
```

```
    public override string ToString()
    {
        return $"({x}, {y})";
    }
}

public class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
        Point p1 = new Point(0, 0);
        Point p2 = new Point(1, 0);
        Point p3 = new Point(1, 1);
        Point p4 = new Point(0, 1);
        Rectangle rect = new Rectangle(p1, p2, p3, p4);


        rect.Print();
        rect.Rotate(new Point(0, 0), 90);
        rect.Print();

        Console.WriteLine("----------");

        Rectangle rect1 = new Rectangle(new Point(1, 1), new Point(1, 3), new Point(3, 3), new
Point(3, 1));
        rect1.Print();
        rect1.Rotate(new Point(0, 0), 90);
        rect1.Print();
        rect1.Rotate(new Point(0, 0), 60);
        rect1.Print();
        rect1.Rotate(new Point(0, 0), 30);
        rect1.Print();
    }
}
```