Projekt Threading

En tråd er en måde at kalde en metode parallelt med andre metoder. Dette kan skabe problemer hvis flere tråden har adgang til den samme data, derfor kan man bruge egenskaber i C#, som fx. **Mutex**, som kan bede andre tråde om at vente, så længe dataen bliver håndteret. En **Monitor** kan sætte en lås på trådens kritiske stykke kode, så den data, som ikke må tilgås af flere tråde på samme tid, bliver låst, med adgang af kun en tråd. En **Semaphore** kan give adgang til flere tråde på én gang. Antallet af tråde som kan få adgang og antallet af tråde som allerede har adgang bliver sat i semaforens constructor.

```
private static int bossHealth = 6;
```

bossHealth er dataen som alle tråde har adgang til.

De Tre Synkronisering Mekanismer

Vi har ikke brugt de tre mekanismer af nogen kritisk årsag, men kun for at få erfaringer med dem alle.

Mutex

```
static Mutex handleDamage = new Mutex();
Thread damage = new Thread(new ParameterizedThreadStart(Dps));
```

Denne tråd, som kører metoden **Dps**, instantieres med navnet **damage.** Den tager et parameter med ind, som kører **dps** fra **Archer** klassens constructor.

```
public Archer(string imagePath, Vector2D startPosition, int dps,
{
    damage.Start(dps);
    myForm = newForm;

}
```

Selve tråden håndteres med en **Mutex: handleDamage**. While loopen for tråden kører så længe **GameWorld.BossHealth** er mindre end 2.000.000.000.
handleDamage.WaitOne() sørger for, at der ikke kan komme andre ind, mens koden kører og lader den næste komme ind, når den releases i **handleDamage.RealeaseMutex().**
tråden sover i 1500 millisekunder, så **Archer** klassen kun angriber hvert 1.5 sekund.
Thread.Sleeps' parameter kan være ændret i release.

```
public static void Dps(object obj)
{
    int dps = (int)obj;
    while (GameWorld.BossHealth <= 2000000000)
    {
        handleDamage.WaitOne();
        GameWorld.BossHealth -= dps;
        myForm.AlternativeClick();
        handleDamage.ReleaseMutex();
        Thread.Sleep(1500);

    }
}
```

Thread Lock

```
static Object thisLock = new Object();
Thread damage = new Thread(new ParameterizedThreadStart(Dps));
```

```
public static void Dps(object obj)
{
    int dps = (int)obj;
    while (GameWorld.BossHealth <= 2000000000)

    {
        lock (thisLock)
        {
            GameWorld.BossHealth -= dps;
            myForm.AlternativeClick();
        }
        Thread.Sleep(500);
    }
}
```

lock bruges ved man laver et object i dette tilfælde **thisLock** instansiere objectet
så laver vi en tråd der hedder **damage** der tager metoden dps med parameteren object eftersom alle ting i c# på bund niveau er objecter caster vi parameteren til en int.
(while løkken er der kun for trådene bliver ved med at angribe, da det er nødvendigt for spillet at enhederne angriber da det eller er et mærkeligt cookie clicker)

så når en tråd går ind til det kritiske område som kunne skabe problemer , fx ved at flere tråde opdaterede livet på bossen samtidig så den kunne fx være i live mens en anden opdaterede så den faktisk var død og ikke burde have kunnet køre anden kode, så låser vi det kritiske område med **thisLock** så kun en tråd kan komme ind ad gangen, og opdatere i dette tilfælde bossens liv.
så beder vi tråden om at sove med thread.sleep så de ikke angriber konstant , så det fungere som cooldown på deres angreb.

Semaphore

```
Thread damage = new Thread(new ParameterizedThreadStart(Dps));
public static Semaphore maxdmg = new Semaphore(0, 5);
```

```
public static void Dps(object obj)
{
    int dps = (int)obj;

    while (GameWorld.BossHealth <= 2000000000)

    {
        maxdmg.WaitOne();
        GameWorld.BossHealth -= dps;
        myForm.AlternativeClick();
        maxdmg.Release();
        Thread.Sleep(1000);

    }
}
```

Man bruger en **Semaphore** til at lukke flere tråde ind af gangen og samtidig styre hvor mange der max kan være der inde og ændre på tingene inde i det kritiske område. I vores spil bruger vi den til vores **Swordman** vær gang du laver en ny **Swordman** bliver hans damage fordobblet, og der bliver oprettet en ny tråd for hans angreb. Så for at holde styr på hvem der får lov til at angribe har vi sat en semafor ind så der max kan være 5 inde og ændre **bossHealth** af gangen.

```
public void SetupWorld()
{
    Swordman.maxdmg.Release(5);
}
```

Hver gang der har været en tråd inde i det kritiske punkt sætter man den til at release dens spot så en ny tråd får mulighed for at komme der ind og gøre det som den skal der.

Delegate

```
class Knight : Unit
{
    private static Form1 myForm;
```

vi erklærer først **Form1 som myForm** så vi har et field at putte referencen til forms så vi kan invoke den fra **Form1.cs**

```
public Knight(string imagePath, Vector2D startPosition, int dps, Form1 newForm) : base(imagePath, startPosition)
{
    damage.Start(dps);
    myForm = newForm;
}
```

så har knight fået parameteren **Form1 newForm** med i constructoren som vi så lægger ind i fielded **myForm** i constructorens body

```
public static void Dps(object obj)
{
    int dps = (int)obj;
    while (GameWorld.BossHealth <= 2000000000)

    {
        lock (thisLock)
        {
            GameWorld.BossHealth -= dps;
            myForm.AlternativeClick();
        }
        Thread.Sleep(500);
    }
}
```

så kører vi **myForm.Alternativeclick()**

```
public void AlternativeClick()
{
    if (GameWorld.BossHealth <= 0)
    {
        Random rand = new Random();

        choosenDragon = rand.Next(1, 10);
        if (prevDragon == choosenDragon)
        {
            choosenDragon = rand.Next(1, 10);
        }


        pictureBox1.Image = Image.FromFile("Sprites/Dragon/dragon" + choosenDragon + ".png");

        pictureBox1.Invoke((MethodInvoker)delegate { pictureBox1.Refresh(); });
        pictureBox1.Invoke((MethodInvoker)delegate { pictureBox1.Visible = true; });

        prevDragon = choosenDragon;

    }
}
```

AlternativeClick() har samme kode som **picturebox1_click** men det virkede kun når vi klikkede på pictureboxen, og ikke når trådene var den der slog bossen ihjel. så derfor

invoker vi **picturebox1** så vi kan kalde den fra andre tråde som set ovenfor med **myForm.AlternativeClick()**
delegate kan give metoder videre til andre metoder som argumenter og her giver vi så **pictureBox1** metoden **pictureBox.refresh()** med som hvis vi gjorde det fra main tråden. det samme med **picturebox1.Visible=true.**

```
        private void pictureBox1_Click(object sender, EventArgs e)
        {
            GameWorld.BossHealth -= GameWorld.PlayerDmg;

            if (GameWorld.BossHealth <= 0)
            {
                Random rand = new Random();

                choosenDragon = rand.Next(1, 10);
                if (prevDragon == choosenDragon)
                {
                    choosenDragon = rand.Next(1, 10);
                }


                pictureBox1.Image = Image.FromFile("Sprites/Dragon/dragon" + choosenDragon + ".png");

                pictureBox1.Refresh();
                pictureBox1.Visible = true;
                //this.PictureBox1.SizeMode = PictureBoxSizeMode.
                prevDragon = choosenDragon;

            }
    }
```

samme kode som **AlternativeClick()** men som man kan se må den nemlig gerne kører metoderne selv som er dem vi har invoked.(ellers er det her billede **overflødigt** men er mere som reference og forståelse senere)