Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatikai Tanszék
Számítógép-alapú rendszerek specializáció

# Rendszer és alkalmazástechnika laboratórium 2.

Jegyzőkönyv

7. mérés

## Refaktorálás, objektum orientált feladat dekompozíció

**Mérést végző hallgatók:** Dámsa Levente EQMJDP
Steinmann György GWK81V

**Mérőcsoport:** Kurzus: Szerda, Csoport: 3

**Mérés időpontja:** 2021.03.03 de

**Mérés helyszíne:** BME AAIT, labor: QB 121/125/127

**Mérésvezető:** Kovács Viktor

### Felhasznált eszközök

| [eszköz neve] | [eszköz típusa] | [gyári v. leltári szám] |
|---|---|---|

# Mérési feladatok

## 1. feladat

---

Feladat rövid megfogalmazása

Snippet szerinti javítások elvégzése

---

Feladat megoldása

A snippet alapján elvégeztük a szükséges módosításokat, a módosításoktól nem tértünk el. Az alábbi lepéseket végeztük

---

Először a Main class-ból szerveztük ki a Turkmite teljes logikáját úgy, hogy a mainben csak a példányositas es a nagy Step metódus maradjon. Igy mar egy osztályként tudtuk kezelni a turkmite-ot.
Utána a “magic konstansok” eltávolítása, es a switch-case szerkezet átalakítása történt. A snippetben megadott (x,y) named tuple helyett először egy dictionary-re gondoltunk, ahol külön deltaX es deltaY lett volna, deltaX-ben 1-es es 3-as kulcs szerepelt volna a megfelelő iránnyal, deltaY-ban pedig 0 es 2. Ezt elvetettük, mert a snippetben található megoldás egyszerűbbnek tűnik.
Ezután a threshold ellenőrzést cseréltük le a snippet alapján, ahol a 199 helyett a kép szelessége es magassága szerepelt, ezzel növelve a rugalmasságot. A színekhez létrehoztuk a Black es White readonly property-t, amivel kikerültek az ideiglenesen létrehozott Vec3b változók.
Ezután a step függvényt bontottuk fel két jól elválasztható részre, ahogy a snippetben szerepel. Először a GetColorUpdateDirection es Move függvénynek neveztük el, de később áteveztük NextDirectionControl-ra, ahol mar nem csak a szint, hanem az irányt is visszaadjuk, a snippet alapján.
Ezután következett az ősosztály létrehozása a snippet alapján. Itt sem tértünk el, az ősosztály tartalmazott minden kiszervezhető elemet.

## 2. feladat

---

Feladat rövid megfogalmazása

3 színű turkmite

---

Feladat megoldása

A snippet alapján létrehoztunk a TurkmiteThreeColor nevű osztályt, amiben szerepelt három szín-Vec3b, es a megfelelő NextDirectionColor felüldefiniálása a logika alapján.Itt meg nem hoztunk létre külön forrást osztályonként, mivel a harmadik feladatnál kényelmesebben lehet egy forráskódban kezelni. Kipróbáláskor kék színű volt a Pirosnak megálmodott harmadik szín, ennek az volt az oka, hogy az opencv BGR leírást használ.

# 3. feladat

Feladat rövid megfogalmazása

Állapotgéppel rendelkező Turkmite készítése

Feladat megoldása

A követelményeket figyelembe véve, es a State design patternt szem előtt tartva az alábbi módosításokkal implementáltuk az állapotgépes Turkmiteot.

Két eseményt szeretnénk kezelni a State-ekben: amikor belepunk az új állapotba, es amikor a Turkmite lekéri az uj directiont es colort. Az új allapot belepéséért az Enter () függvény felel, aki reseteli a State belső változóit, például számlálókat. Az új irány es szín frissítéséért a HandleUpdate() függvény szerepel, ami megkapja az aktuális szint, es állapottól függően beállítja es frissíti a stateMachine-t. Ennek a megoldásnak az ajánláshoz képest több előnye is van.

A State leszármazottak fognak teljes mertekben felelni a state-ek lekezelésében. Ezenkívül a Red(), White(), Black() metódus hátránya az, hogy nem tud rugalmasságot adni abban az esetben, ha esetleg a State logika valamiért változna. Ezért úgy döntöttünk, hogy a handleState() függvény jobb megoldás lenne. Igy a hívónak csak annyi feladata lesz, hogy a handler függvényt meghívja, nem kell tudnia a belső állapotokról, eseményekről, lényegében egy esemény van: lépni kell.

Ezentúl az Enter függvény kezeli a State állításokat is, vagyis az Enter() függvény reseteli magat, majd beállítja a currentState-et. Igy a state váltáshoz csak a megfelelő State Enter() függvényét kell meghívnia. Ennek a megoldásnak az a hátránya, hogy külső state állitás esetén az Enter függvényt is meg kell hívni.

StateTurkmite osztaly:

```
class StateTurkmite : TurkmiteBase
{
    public StateBase currentState;

    readonly public StateB stateB;
    readonly public StateC stateC;
    readonly public StateA stateA;
    public override int IterationCount => 800000;


    public StateTurkmite(Mat image) : base(image)
    {
        stateA = new StateA(this);
        stateB = new StateB(this);
        stateC = new StateC(this);
        currentState = stateA;

    }

    protected override (Vec3b newColor, int deltaDirection)
NextDirectionColor(Vec3b currentColor)
    {
        return currentState.HandleUpdate(currentColor);
    }

}
```

**StateBase** ososztaly:
A színeket csak a StateBase-ben kell definiálva, mivel csak o kezeli a színeket, igy a StateTurkmite osztályból ki lehet torolni őket.

```
abstract class StateBase
{

    public StateTurkmite stateTurkmite;
    readonly protected Vec3b black = new Vec3b(0, 0, 0);
    readonly protected Vec3b white = new Vec3b(255, 255, 255);
    readonly protected Vec3b red = new Vec3b(0, 0, 255);
    public StateBase(StateTurkmite stateTurkmite)
    {
        this.stateTurkmite = stateTurkmite;
    }
    public abstract void Enter();
    public abstract (Vec3b newColor, int deltaDirection)
HandleUpdate(Vec3b currentColor);
}
```

**StateA** ősosztály:
A *stateA* osztályhoz tartozik egy colourCounter változó, ami az Enter()-ben resetelődik, es az UpdateHandle függvényben pedig szerepel a logika. Itt látható az elteres, miszerint a megkapott Turkmite osztály stateB Enter() függvényével vált állapotot. A colourCounter nullázása nem feltétlenül szükséges a konstruktorban, viszont, ha nem az A state lesz a kiinduló allapot, akkor a program igy is helyesen fog futni.

```
    private int colourCounter;
    public StateA(StateTurkmite st) : base(st)
    {
        colourCounter = 0;
    }

    public override void Enter()
    {
        stateTurkmite.currentState = stateTurkmite.stateA;
        colourCounter = 0;
    }
    public override (Vec3b newColor, int deltaDirection)
HandleUpdate(Vec3b currentColor)
    {
        /* egyenesen megy tovabb */
        if (currentColor == black)
        {
            colourCounter++;
        }
        if (colourCounter == 3)
        {
            /* enter next state: B*/
            stateTurkmite.stateB.Enter();
            return (white, 2);
        }
        return (currentColor, 0);
    }
}
```

**StateB**:

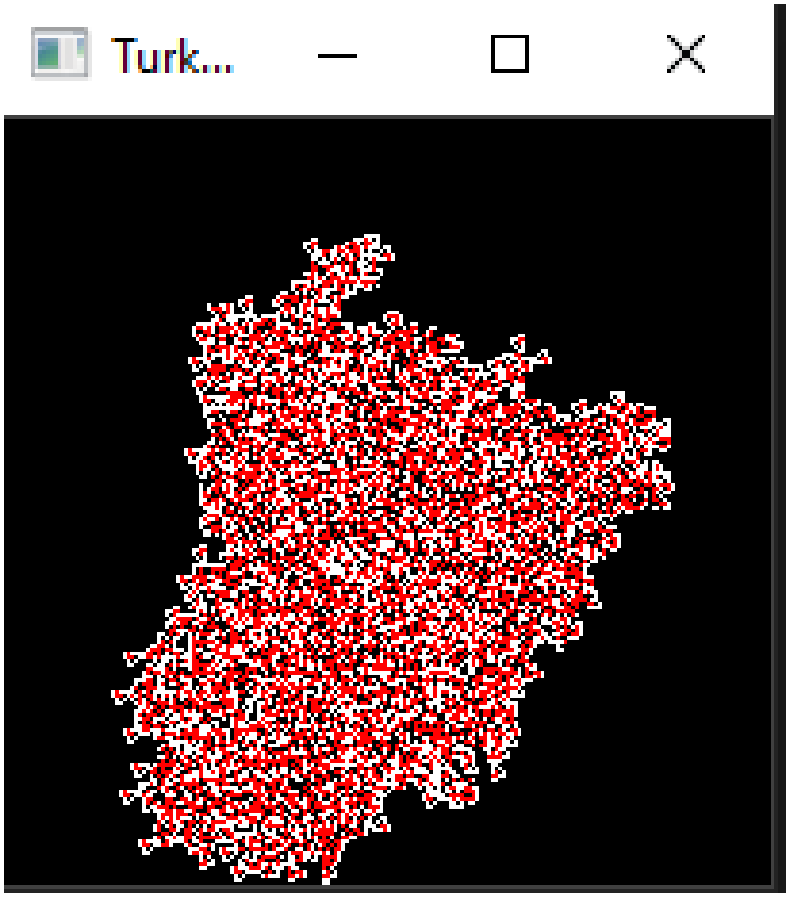
```
class StateB : StateBase
   {
       public StateB(StateTurkmite st) : base(st)
       {
       }
       public override void Enter()
       {
           stateTurkmite.currentState = stateTurkmite.stateB;
       }
       public override (Vec3b newColor, int deltaDirection)
HandleUpdate(Vec3b currentColor)
       {
           if (currentColor == black)
           {
               return (white, 1);
           }
           if (currentColor == white)
           {
               return (red, -1);
           }
           stateTurkmite.stateC.Enter();
           return (black, 0);
       }
```

**StateC**:

StateC eseten szükséges volt egy counter, illetve a StateTransition() függvény létrehozása, amiben lekezeljük a következő state-re való ugrást. Igy a HandleUpdate kezelhető méretű maradt.

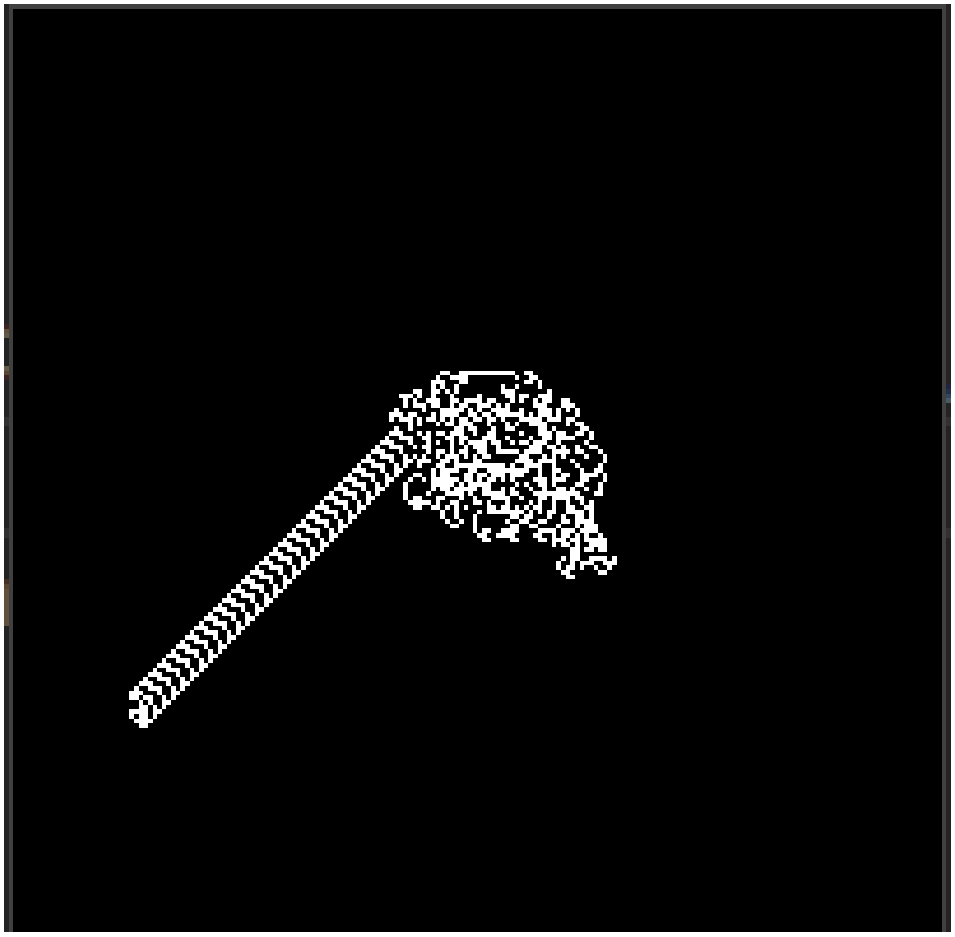
```
class StateC : StateBase
   {
       private int counter;
       public StateC(StateTurkmite st) : base(st)
       {
           counter = 0;
       }

       public override void Enter()
       {
           counter = 0;
           stateTurkmite.currentState = stateTurkmite.stateC;
       }
       public override (Vec3b newColor, int deltaDirection) HandleUpdate(Vec3b currentColor)
       {
           counter++;
           CheckStateTransition(currentColor);
           if (counter == 1)
           {
               return (red, 0);
           }
           if (currentColor == black)
           {
               return (white, 1);
           }
           if (currentColor == white)
           {
               return (red, 0);
           }
           else
           {
               return (black, 3); // turn left
           }
       }
       private void CheckStateTransition(Vec3b currentColor)
       {
           if (counter == 5)
           {
               if (currentColor == red)
               {
                   stateTurkmite.stateB.Enter();
               }
               else
               {
                   stateTurkmite.stateA.Enter();
               }
           }
       }
   }
```

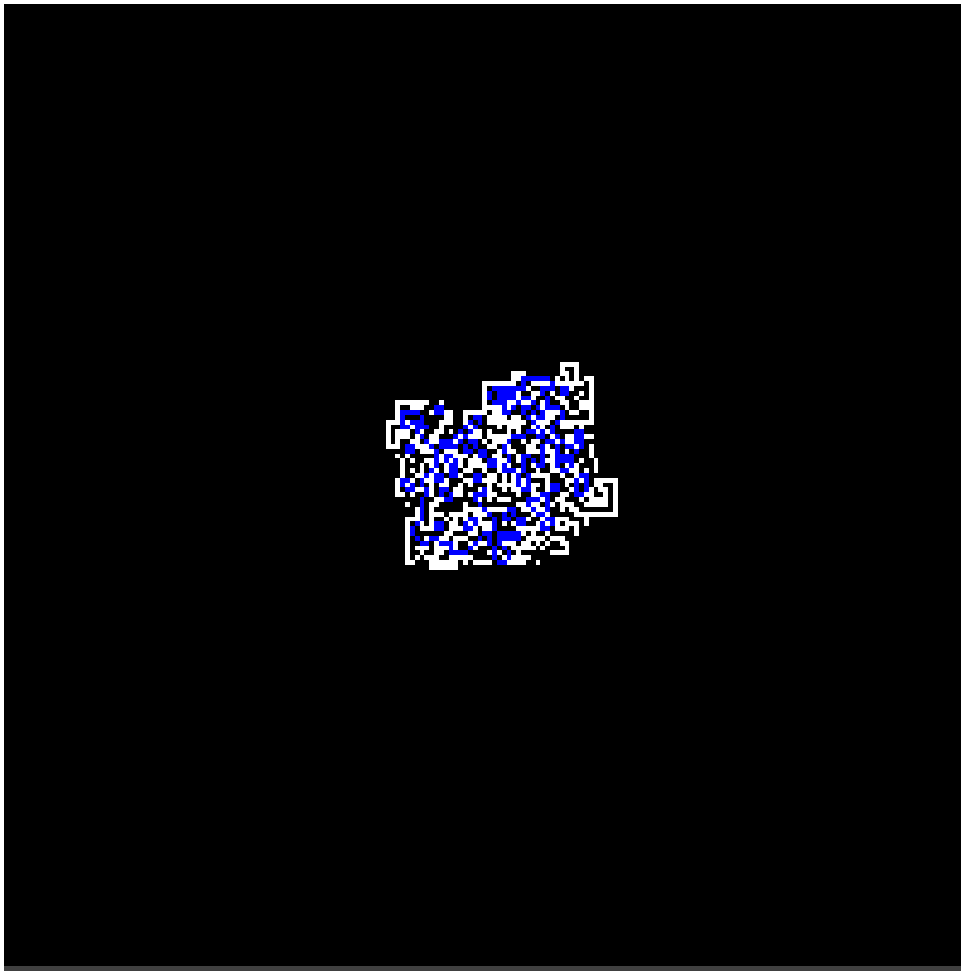
Eredményként az alábbi ábrát kaptuk 800_000 iterációnál:

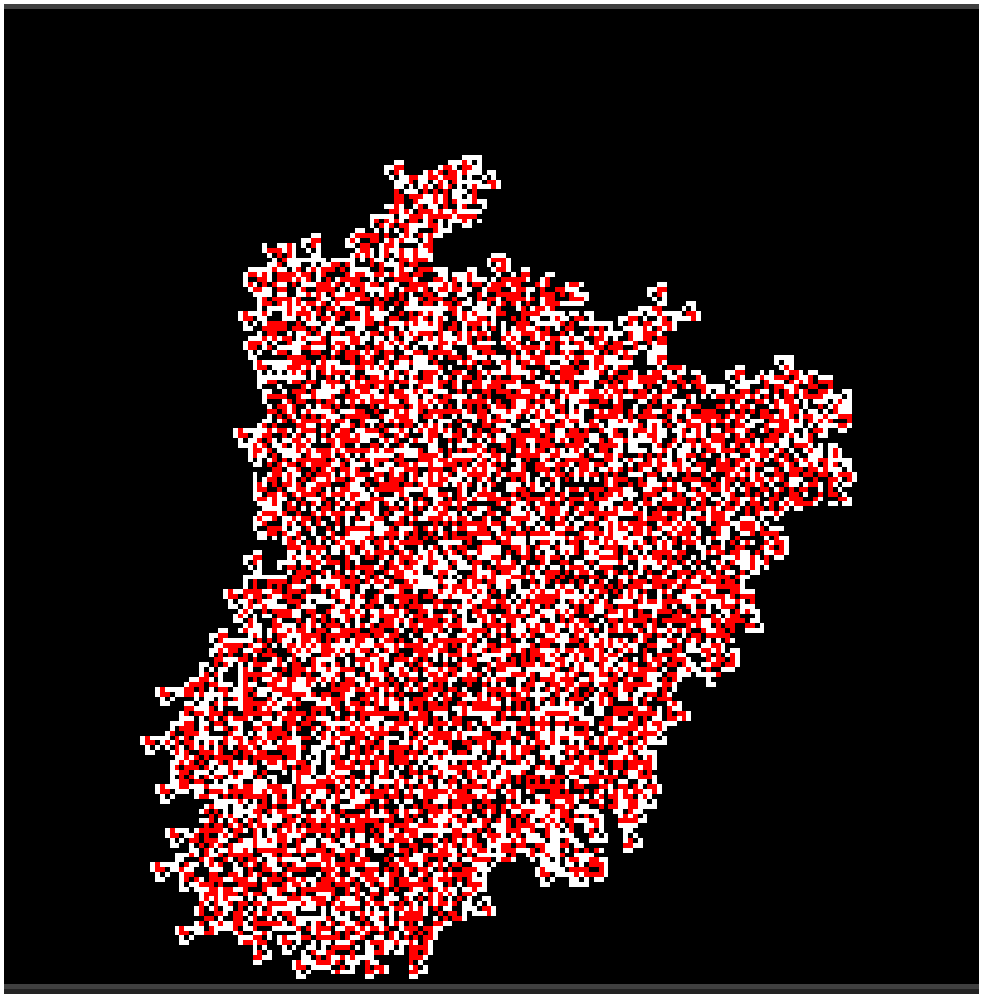



## Összefoglalás

A merést először visual studio nélkül kezdtük el, igy az elején sok syntax error volt, amit a d6e507c0558c2d88b24359a00fafe55c57728d05 commit hash után tettünk futtathatóvá. Az elkészült original,ThreeColour, State kepei:
A ThreeColour eseten nem lett meg javítva a piros szín, csak a StateTurkmite-nál.







Továbbfejlesztési lehetőségként az allapotok váltásához a sok if helyett be lehetne tenni dictionary-ket, amelyek szín – függvény kulcs érték párok lennének.

Ezenkívül a Program.cs-ben a for ciklust helyettesíteni lehetne egy Wrapper classal, amit mondjuk TurkmiteSimulator-nak lehetne nevezni, aminak csak egy Start függvénye lenne, es ki lehetne egészíteni esetleg statisztikával, loggolassal stb.

# Függelék

A repository elérhetősége: https://github.com/damsalevente/start