ELEC-C8203 Automaatiojärjestelmät 2

Automaation tietotekniikka labra 3

Oppimistavoitteet

- Tuotantoresurssien orkestrointi sekä tuotantosolun ohjauksen, että olio-ohjelmoinnin näkökulmasta
- Edellisissä harjoituksissa opittujen asioiden itsenäinen soveltaminen aiempaa laajempiin tehtäviin
- Harjoituksen päätteeksi opiskelijalla tulisi olla tietotekniset perusvalmiudet ryhtyä tutustumaan automaatiopyramidin ylempien kerrosten ohjelmistotekniikkaan, eli:
    - kyky suunnitella ja toteuttaa useamman luokan sovellus Javalla tai Pythonilla (tässä tehdään Javalla, mutta jos asiat on sisäistetty niin vastaavan tekeminen Pythonilla pitäisi onnistua kun googlaa vastaavat Python kielen syntaksit)
    - kyky hyödyntää APIa (tässä jMonkeyEngine API sekä Javan omia APIa lähinnä ArrayList tapauksessa)

Tehtävänanto

Tehdään ensin legoille varastopaikka kokoonpanoaseman taakse siten että robotti ylettää sinne. Tehdään varastosta älykäs siten, että se tietää minkä värisiä legoja sillä on ja voi vastata, kun joku kysyy, että mistä koordinaateista löytyy tietyn värinen lego. Luodaan LegoBuffer luokka mygame-pakettiin (täydennä ??? kohdat):

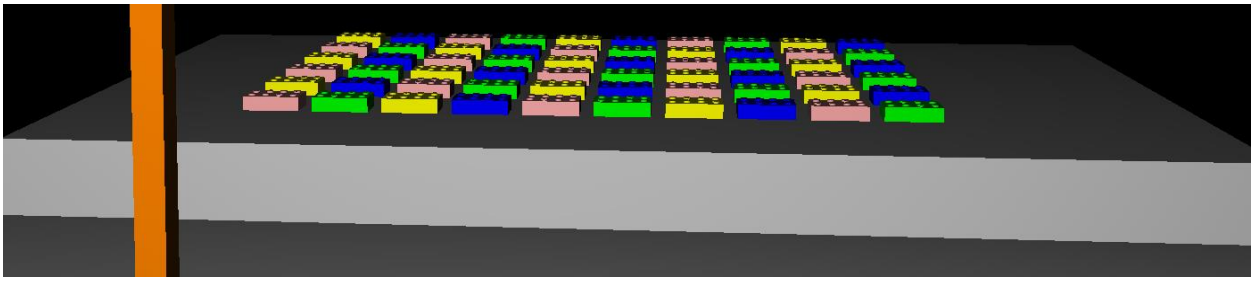
```
public class LegoBuffer {
    private Box box;
    private Geometry geom;
    private float surfaceHeight;
    ArrayList<Lego> legos = new ArrayList(500);
    float x;
    float z;
    private float legoSpacingX = 2;
    private float legoSpacingZ = 2;
    int rowSize;
    int columnSize;

    // luo varastotason koordinaatteihin x=xOffset, z=zOffset (y-korkeus siten että se
    // on solun lattialla. Legoja on riveissa ja sarakkeissa, joihin mahtuu ”rowSize”
    // * ”columnSize” legoja
    public LegoBuffer(AssetManager assetManager, Node rootNode, float xOffset,
                      float zOffset, int rowSize, int columnSize) {
        // tallennetaan parametrien arvot olion yllämääriteltyihin muuttujiin, jotta
        // ne olisivat käytössä sen jälkeen kun tämä konstruktori metodi on suoritettu
        this.rowSize = rowSize;
        this.columnSize = columnSize;
        x = xOffset;
        z = zOffset;

        // buffer on box jonka yExtent on:
        float yExtent = 7;
```

```
        // luo ColorRGBA.LightGray värinen box jonka koko on (16f, yExtent, 8f)
        // luo sille geometria ja liitä se rootNodeen
        ???

        // tason pinnan y koordinaatti lasketaan Main.floorHeight + yExtent avulla
        surfaceHeight = ???
        // laitetaan varastotaso siten että pohja on tuotantosolun lattian korkeudella
        geom.setLocalTranslation(x, ???, z);

        String colorLego = "red"; // punainen lego tulee vain jos koodissasi on bugi
        // laitetaan tasolle rowSize*columnSize legoa. Vuorotellen eri värisiä. Muista
        // Lego konstruktori hyväksyy nämä: ”yellow”, ”blue”, ”pink”, ”green”
        for (int i = 0; i<(rowSize*columnSize/4); i++) {
            for (int j=0; j<4; j++) {
                // asetetaan colorLego arvo, vuorotellen eri väri
                ???
                Lego lego;
                // luodaan lego, lisätään se legos ArrayListiin, liitetään rootNodeen
                ???
            }
        }
        for(int i=0; i<(rowSize*columnSize); i++) {
            legos.get(i).node.setLocalTranslation(getLegoCenterLocation(i));
        }
    }

    // legon x koordinaatti suhteessa LegoBuffer keskipisteeseen. ”legos” listan eka
    // lego on paikassa rowIndex=0 columnIndex=0. Listan toka lego on paikassa
    // rowIndex=0 columnIndex=1 jne
    private float xCoord(int index) {
        int rowIndex = index % rowSize;
        return (rowIndex - rowSize/2) * legoSpacingX;
    }

    // legon z koordinaatti suhteessa LegoBuffer keskipisteeseen. ”legos” listan eka
    // lego on paikassa rowIndex=0 columnIndex=0. Listan toka lego on paikassa
    // rowIndex=0 columnIndex=1 jne
    private float zCoord(int index) {
        ???
    }

    // palauttaa legos listan “index” kohdan legon (keskipisteen) koordinaatit
    // palauta maailma-koordinaatit eli huomioi x, z ja surfaceHeight
    // 0.2f on y-etäisyys pöydän pinnasta legon keskipisteeseen
    // käytä xCoord() ja yCoord() metodeja
    private Vector3f getLegoCenterLocation(int index) {
        return new Vector3f(x+xCoord(index), surfaceHeight + 0.2f, z+zCoord(index));
    }

    public float getSurfaceHeight() {
        return surfaceHeight;
    }
}
```

Luo yksi instanssi tästä luokasta Main.javassa seuraavasti (mieti mihin kohtaan tiedostoa tämä rivi kannattaa laittaa):

```
legoBuffer = new LegoBuffer(assetManager, rootNode, 5, -29, 10, 6);
```

Vinkki: zCoord() toteutuksessa katso mallia xCoord():sta. Kyseiset metodit antavat legojen varastopaikkojen koordinaatit LegoBuffer-varastotason pinnalla. Menettele niin että lopputulos näyttää tältä kun ajat ohjelman:



Seuraavaksi pitää ohjelmoida robotti hakemaan legoja ja tuomaan ne oikeaan paikkaan kokoonpanoasemalle. Tehtävä jakaantuu neljään osaan:

1. reitinsuunnittelu (APP)
2. pitää tietää mihin kohtaan kokoonpanoasemaa lego toimitetaan
3. pitää tietää varastopaikan koordinaatit, mistä löytyy tietynvärinen lego
4. ASP: jossain pitää koordinoida koko sekvenssi, jolla haetaan kaikki legot haluttuihin paikkoihin

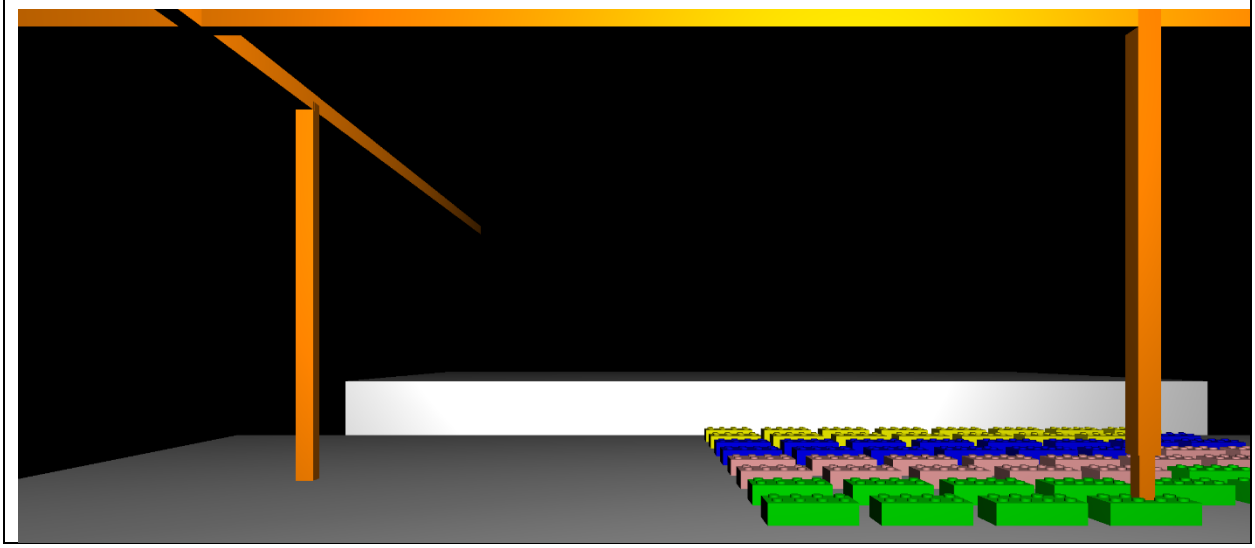
Teoria
Kuten aiemmin mainittiin, ASP (Assembly Sequence Planning) on algoritmi, joka suunnittelee missä järjestyksessä kokoonpantavat osat käsitellään. Usein siinä pitää huomioida, mitkä kappaleet kiinnitetään toisiinsa ja mikä järjestys on tarpeen jotta vältetään törmäykset. Ongelmaan tuo huomattavaa lisähaastetta, että se on kolmiulotteinen. Helpoimpia tapauksia ovat kaksiulotteiset kokoonpanotehtävät kuten elektroniikkateollisuuden kokoonpanotehtävät, joissa komponentteja laitetaan piirilevylle. Tämän harjoituksen tehtävä muistuttaakin elektroniikkakokoonpanoa, sillä legot vain toimitetaan haluttuihin paikkoihin. Harjoituksen tehtävä myös muistuttaa 3D kokoonpanon esivaihetta, missä koottavat kappaleet toimitetaan kokoonpanoasemalle kokoonpanorobotin ulottuville. Tällöin ei ole kysymyksessä ASP.

Suunnitellaan ohjelmisto seuraavasti:

1. APP tehdään entiseen tapaan AssemblyStation luokkaan. Korvataan `initTestMove()`.
2. Lisätään AssemblyStation luokkaan metodi `slotPosition(int slot)`
3. Lisätään LegoBuffer luokkaan metodi `giveLego(String color)`, joka palauttaa halutun värisen Lego olion. Oliossa puolestaan on legon senhetkisen paikan 3D koordinaatit, joten niitä voi käyttää APP:ssa.
4. ASP tehtäisiin tuotantosolun tasolla. Aiemmin päätettiin, että kun sovellus on pieni ja yksinkertainen, niin ei tehdä erillistä luokkaa solua varten vaan pannaan koodit Main luokkaan.

Tehtävänanto

Siirrä kaikki legot kokoonpanoasemalle ja lajittele värin mukaan. Ensin keltainen, sitten sininen, pinkki ja vihreä, Kts kuva:



LegoBuffer luokka

```
// APP tarvitsee legonyläpinnan koordinaatin, johon robotti tuo työkalunsa alapinnassa
// olevan 'tooltip' pisteensä
private Vector3f getLegoTopLocation(int index) {
    return new Vector3f(x+xCoord(index), surfaceHeight + 0.4f, z+zCoord(index));
}

// palauttaa Lego olion joka on halutun värinen tai null jos tällaista legoa ei ole
// päivitä legoColor Lego luokkaan ja konstruktoriin
// päivitä location Lego luokkaan
public Lego giveLego(String color) {
    Lego lego = null;
    // luupataan läpi kaikki bufferin legoslotit
    for(int i=0; i<(rowSize*columnSize); i++) {
        lego = ???
        if(lego != null) {
            if(???) {
                lego.location = getLegoTopLocation(i);
                legos.set(i, null);
                return lego;
            }
        }
    }
    return null;
}
```

AssemblyStation luokka

```
float legoSpacingX = 2; // legojen slottipaikkojen etäisyys x-suuntaan
float legoSpacingZ = 2; // legojen slottipaikkojen etäisyys z-suuntaan


// kokoonpanoasemalla on slotteja, joiden indeksi on kokonaisluku
// tämä palauttaa slotin 3D koordinaatit
public Vector3f slotPosition(int slot) {
    // vain osa asemasta on varattu tähän tarkoitukseen. Sen koko on 16x12
    int rowSize = (int)((16)/legoSpacingX);
    int columnSize = (int)((12)/legoSpacingZ);
    int rowIndex = slot % rowSize;
    float xOffset = (rowIndex-1) * legoSpacingX;
    int columnIndex = slot / rowSize;
    float zOffset = (columnIndex + 2) * legoSpacingZ;
    float yOffset = 0.4f; // legonyExtent
    // 'x' ja 'z' on float muuttujia, joihin on tallennettu konstruktorin xOffset/zOffset
    // laske 'surfaceHeight' konstruktorissa
    return new Vector3f(x + xOffset, surfaceHeight+yOffset, z + zOffset - 12);
}


// APP kohteeseen lego.location
// sama idea kuin edellisen harjoituksen initTestMove()
public void initMoveToLego(Lego lego) {
    ???
}


// APP kohteeseen destination
public void initMoveToStation(Lego lego, Vector3f destination) {
    assemblyArm.nodeToolTip.attachChild(lego.node); // muuten lego ei lähde mukaan
    // nyt legon noden sijainti pitää määritellä nodeToolTip paikallisissa
    // koordinaateissa. lego.node.setLocalTranslation(0,0,0) laittaisi legon
    // keskipisteen tooltipin keskipisteeseen
    // vinkki: tooltipin yExtent = 0.4f ja legon yExtent = 0.2f
    ???
    // sitten tehdään APP kohteeseen "destination"
    ???
}
```

Main luokka

HUOM: Poista initTestMove() kutsu.

```
boolean freeze = false; // debug tarkoituksiin - laita true siinä kohdassa koodia
                        // mihin haluat robotin pysähtyvän
boolean moving = false; // true kun robotti liikkuu
boolean goingToLego = false; // true kun mennään hakemaan legoa bufferista
Lego lego;
int slotIndex = 0; // kokoonpanoaseman slot
final int numColors = 4; // final on sama kuin C-kielen const
```

```
int colorIndex = 0; // “colors” (kts alla) listan indeksi
// listan perusteella tiedetään missä järjestyksessä lajitellaan värin mukaan
ArrayList<String> colors = new ArrayList(numColors);


public void simpleInitApp() {
    colors.add("yellow");
    // lisää muut värit tehtävänannon mukaisessa järjestyksessä
    ???
}


public void simpleUpdate(float tpf) {
    if(!freeze && moving) {
        moving = station.move();
    }
    if(!moving && !freeze) {
        // moving=false tarkoittaa että saavuttiin reitin päähän, joten on 2 tapausta:
        // otetaan lego mukaan tai jätetään se
        if(goingToLego) { // otetaan lego mukaan
            // nyt ollaan bufferilla sen legon kohdalla mikä otetaan mukaan
            // v:hen laitetaan kokoonpanoaseman slot numero ”slotIndex” koordinaatit
            Vector3f v = ???
            slotIndex++;


            // suoritetaan APP kohteeseen v
            ???
            goingToLego = false;
            moving = true;
        } else { // jätetään lego tähän
            if(lego != null) { // käynnistyksen yhteydessä tätä koodia ei suoriteta
                // lego on nyt toimitettu oikeaan paikkaan kokoonpanoasemalle
                // otetaan paikka talteen ennen kuin irrotetaan noodi
                Vector3f loc = lego.node.getWorldTranslation();
                // irrota legon node tooltipin nodesta
                // (tämä on pitkä rimpsu jossa käytetään monen olion nimeä
                ???
                lego.node.setLocalTranslation(loc);
                // legon node ei ole nyt kiinni missään nodessa, joten se ei tule
                // näkyviin ennen kuin korjaat asian
                ???
            }
            // haetaan bufferista seuraava lego, jonka väri on: colors.get(colorIndex)
            // eli päivitä muuttujan ’lego’ arvo
            ???
            moving = true;
            if (lego == null) {
                // bufferissa ei ole enempää tämänvärisiä legoja
                colorIndex++;
                if(colorIndex >= numColors) {
                    // kaikki legot on siirretty
                    freeze = true; // tämän jälkeen mitään ei tapahdu
                } else {
                    // haetaan bufferista seuraava lego, jonka väri on:
```

```
                    // colors.get(colorIndex)
                    ???
                }
            }
            if(!freeze) {
                station.initMoveToLego(lego);
            }
            goingToLego = true;
        }
    }
}
```

Reflektointi

Teoria
Olio-suunnittelussa ei ole yhtä oikeaa ratkaisua, kun mietitään mitä luokkia meillä on ja mitä toiminnallisuutta sijoitetaan mihinkin luokkaan. Hyvä suunnittelu vähentää *riippuvuuksia (dependency)* luokkien välillä, siten että: • muutos yhteen luokkaan toivottavasti ei edellytä muutoksia muihin luokkiin • ulkopuolisen on helppo käyttää luokkaa lukemalla attribuuttien ja metodien dokumentaatio ilman että on tarpeen tietää miten ne on toteutettu

- Olet suorittanut vähintään yhden ohjelmointikurssin, mutta onko sinulle opetettu ohjelmistosuunnittelua?
- `Main.simpleUpdate()` koodi olisi myös voitu panna johonkin AssemblyStationin metodiin, jota kutsuttaisiin `Main.simpleUpdate():sta.`
    - Miten hyvin tämä olisi toiminut nykyisessä tilanteessa?
    - Miten *ylläpidettävä (maintainable)* ratkaisu tämä olisi, jos jatkossa varastopaikka olisi kauempana kokoonpanoasemasta niin että kokoonpanoaseman robotti ei ylettäisi siihen, vaan legot toisi liikkuva robottialusta (AGV – Automatic Guided Vehicle), joka parkkeeraisi kokoonpanoaseman viereen, niin että kokoonpanoaseman robotti voisi noukkia legot siitä?