

Maze3D - Remote Mobile Control

by CONSTANTIN Catalin, TIBA Daniela

1. Prezentare generala

Maze3D este un labirint construit 3D, in care utilizatorul se poate deplasa utilizand tastele sageti. Deplasarea prin labirint se poate face deasemenea si prin controlul aplicatiei de pe un telefon mobil. Labirintul consta intr-un sir de incaperi, cu obiecte (sfera, thorus) in interiorul acestora.

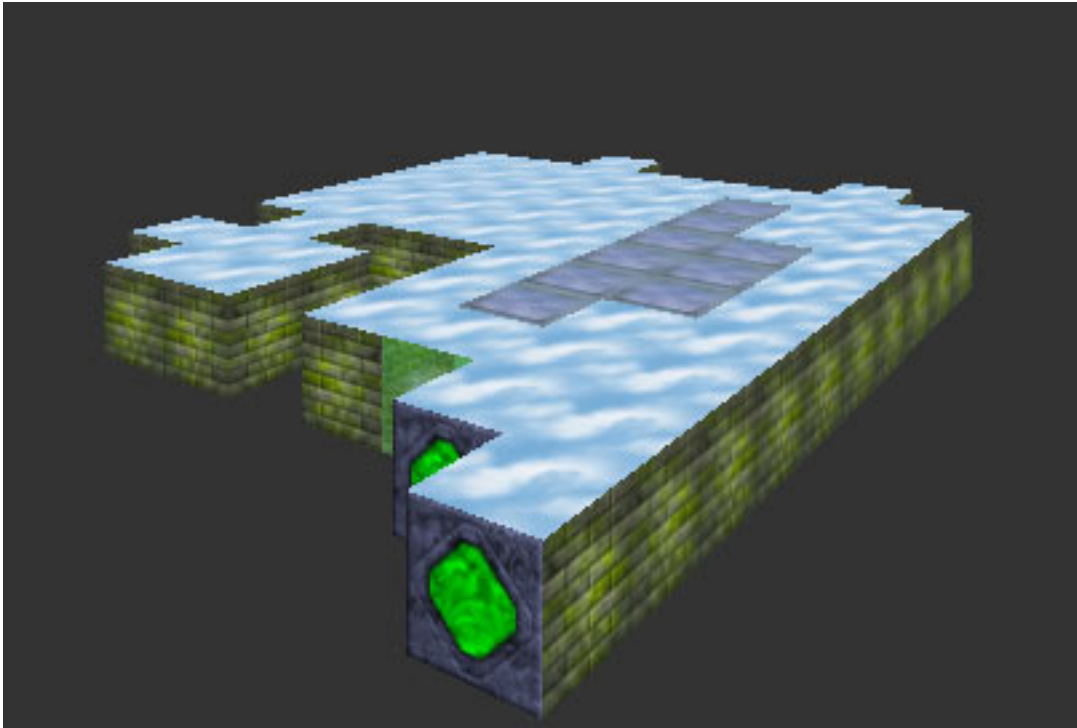
Aplicatia a fost scrisa in Java, fiind compilata cu ajutorul Ant. Pentru realizarea graficii 3D am folosit biblioteca xith3D. Aceasta biblioteca este construita avand la baza functiile de grafica 3D disponibile in Jogl, varianta pentru Java a bibliotecii C OpenGL.

Nota:

Xith3D se poate downloada de la adresa: www.xith3d.org.

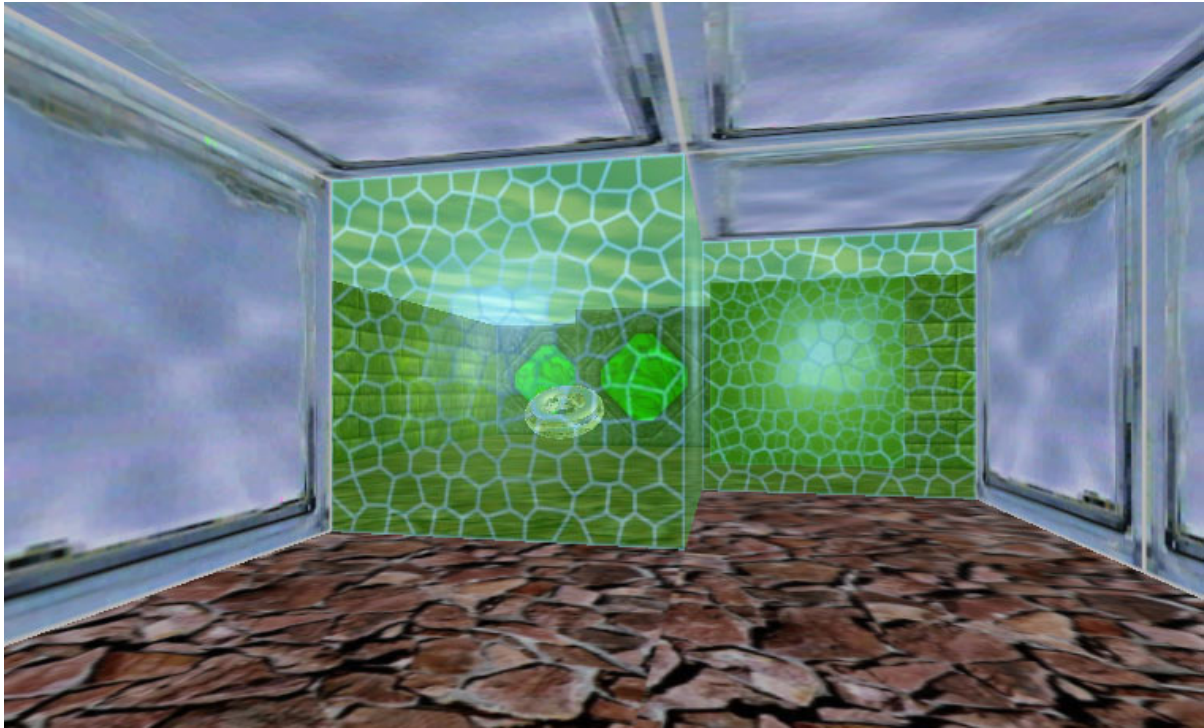
1.1. Controlul aplicatiei de la tastatura

Utilizatorul se poate plimba prin labirintul creat prin intermediul tastelor: stanga, dreapta, inainte, inapoi. Modificarea (cresterea sau diminuarea) vitezei de deplasare prin camera se face cu tastele +, si respectiv -. La lansarea aplicatiei utilizatorul se va gasi la pozitia (0,0,0).



Labirintul vazut de sus

Prin pereti, cat si prin diversele obiecte aflate in camera nu se va putea trece, acestea fiind suprafete blocante. Unii dintre pereti sau obiecte pot fi transparenti, putandu-se vedea de partea cealalta, insa nu se va putea intra prin ei.



Pereti transparenti

In poza de mai sus este prezentata o parte din labirint, si se poate observa proprietatea anumitor pereti de a fi partial transparenti, in timp ce altii sunt complet opaci.

1.2. Controlul remote al aplicatiei

Aplicatia poate fi controlata si prin intermediul unui telefon mobil compatibil Java pe care sa ruleze partea de aplicatie destinata mobilelor.

Pentru detalii va rugam sa cititi mai jos in sectiunea de implementare.



Nokia 7210 Maze3D Mobile Client - preview

2. Implementarea aplicatiei

2.1. Definirea labirintului

Un labirint este format dintr-o multime de celule. Celulele, texturile, tipul lor default, caracteristicile fiecărei in parte, sunt citite dintr-un fisier .xml din directorul data (world.xml).

2.1.1. Formatul fisierului world.xml

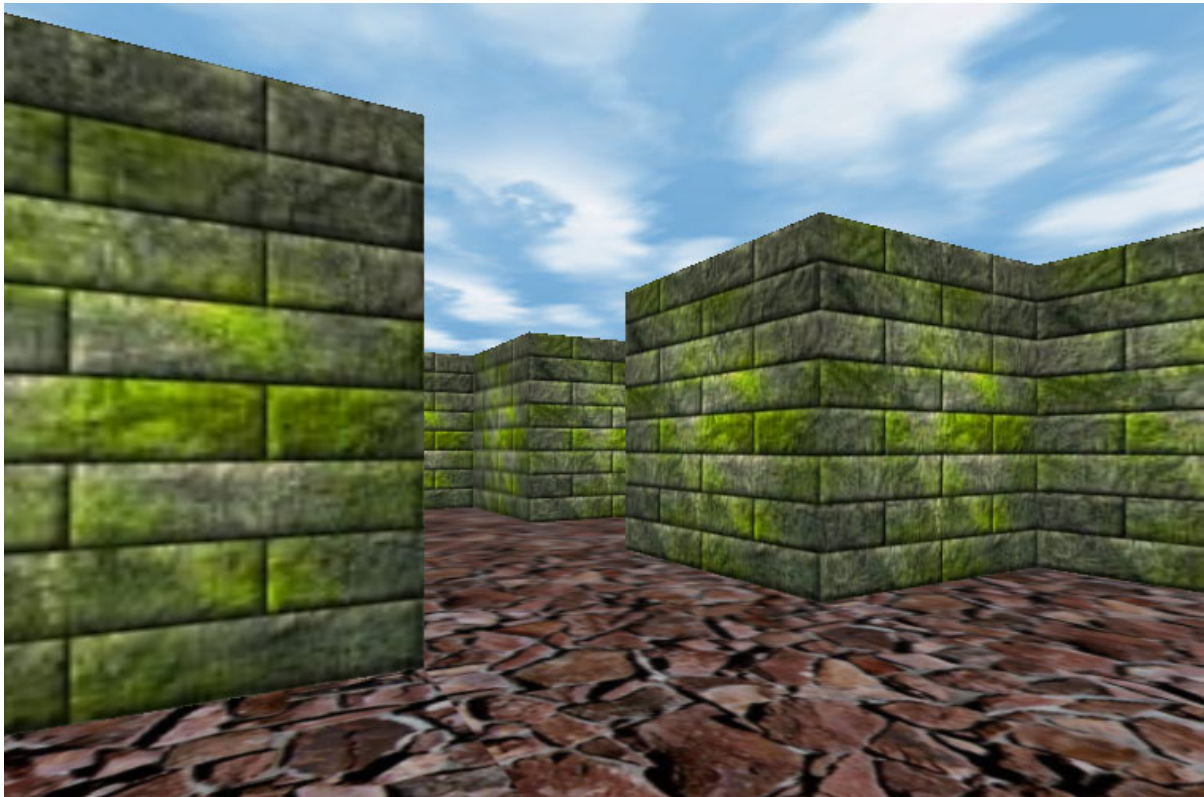
Fisierul este in format .xml are o forma bine definita.

Tagul nod este world. Apoi urmeaza definirea setarilor (settings). La inceputul fisierului sunt specificate tipurile de texturi care pot acoperi diversi pereti ai celulelor. O textura este definita prin id-ul sau, care este un string si reprezinta numele texturii si un filename care contine numele fisierului imagine si directorul in care acesta se gaseste. In continuare este prezentata partea din fisier in care sunt definite texturile.

```
<textures>
  <texture id="perete" filename="data/walls/w1.jpg" />
  <texture id="podea" filename="data/podea.jpg" />
  <texture id="tavan" filename="data/tavan.jpg" />
</textures>
```

```
<texture id="tavan1" filename="data/tavan1.gif" />
<texture id="geam" filename="data/geam.jpg" />
<texture id="outside" filename="data/outside.jpg" />
<texture id="piatra" filename="data/walls/w3.gif" />
<texture id="perete1" filename="data/walls/w5.jpg" />
</textures>
```

In poza urmatoare sunt prezentate diverse texturi aplicate pe diversi pereti ai unor celule.



Texturi diverse

Dupa definirea texturilor ce vor fi folosite pentru pereti si pentru obiectele continute se trece la definirea unei celule a labirintului. O celula este definita prin peretii sai, prin dimensiunile sale (width si height) si prin pozitia la care se gaseste. Un perete are urmatoarele caracteristici: tipul sau, textura, daca este prezent, daca este transparent si tipul de normala pe care o aplicam la suprafata sa.

O celula default va avea prezenti peretii pentru podea si pentru tavan, avand texturile podea, respectiv tavan, cu blending false. Restul peretilor, stanga, dreapta, fata, spate fiind absenti. In codul inserat in continuare este continuata definirea celulei default pentru aplicatia noastra.

```

<cell width="5" height="5" y="0">
  <wall type="podea" texture="podea" present="true"
    blending="false" normaltype="0"/>
  <wall type="tavan" texture="tavan" present="true"
    blending="false" normaltype="0"/>
  <!-- default peretii sunt absenti -->
  <wall type="stanga" present="false" texture="perete"
    blending="false" normaltype="1"/>
  <wall type="dreapta" present="false" texture="perete"
    blending="false" normaltype="2"/>
  <wall type="fata" present="false" texture="perete"
    blending="false" normaltype="3"/>
  <wall type="spate" present="false" texture="perete"
    blending="false" normaltype="4"/>
</cell>

```

In continuare fisierului este descrisa in parte fiecare celula care compune labirintul. Datorita acestui mod de descriere a unui labirint, forma acestuia se poate schimba cu usurinta. O celula pastreaza caracteristicile celulei default daca in definitia sa nu se specifica o alta valoare pentru unul din attributele sale. De exemplu, celula din exemplul urmator are in plus fata de o celula standard prezenti peretii stanga, spate si dreapta, iar cel din spate are o textura diferita de cea default.

```

<cell x="0" z="0">
  <wall type="stanga"/>
  <wall type="spate" texture="piatra"/>
  <wall type="dreapta"/>
</cell>

```

Spre deosebire de exemplul de mai sus, celula urmatoare

```
<cell x="1" z="7"/>
```

nu are nici un perete in plus fata de cea default (podea si tavan) si se gaseste la pozitia x=1 ,z=7 si y=0 (din default).

O alta caracteristica a celulelor unui labirint este prezenta unui obiect (care poate fi orice, spre exemplu o sfera sau un thorus) in interiorul sau. Acest lucru se specifica in world.xml prin existenta unui extrarenderer la anumite celule.

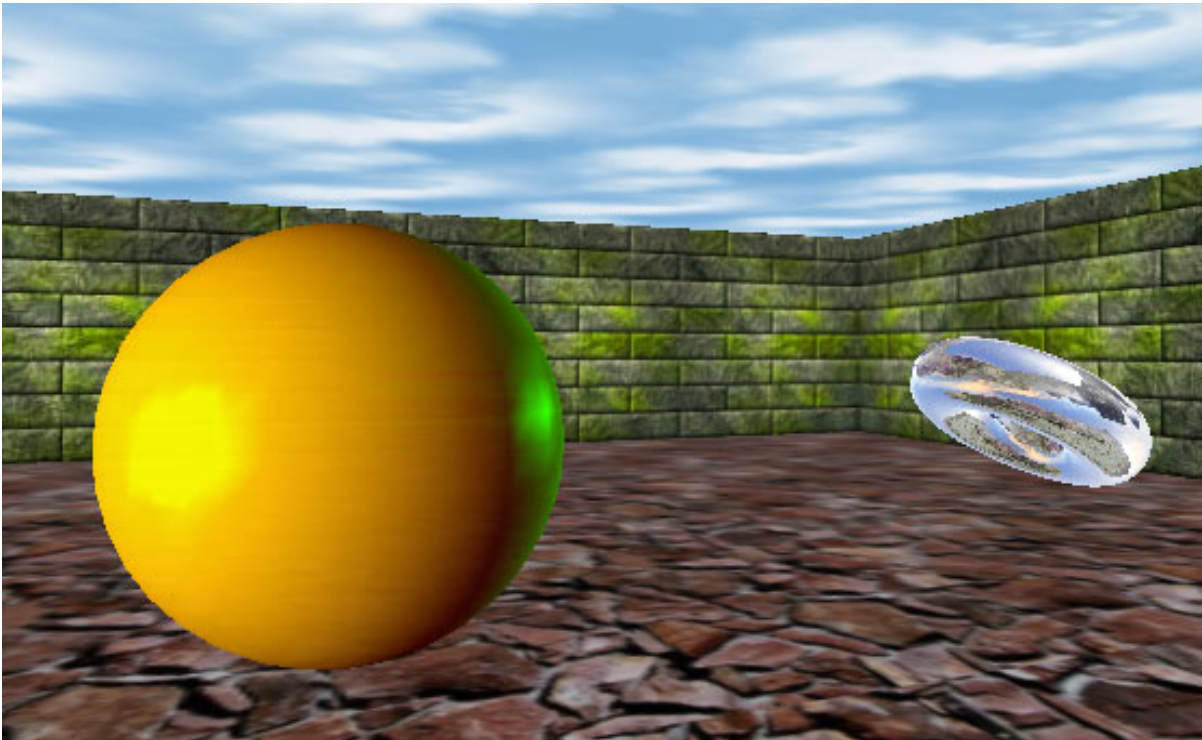
Spre exemplu, prin:

```
<cell x="1" z="2"
```

Maze3D - Remote Mobile Control

```
extrarender="ro.dazoot.maze3d.renderplugins.Thorus" />
```

se specifica faptul ca celula aflata la pozitia $x=1$, $z=2$ are in interiorul sau un obiect definit de tip `Thorus`.



Celule avand in interiorul lor o sfera si un thorus

2.1.2. Citirea si parsarea fisierului de definire world.xml

Citirea si parsarea fisierului .xml se face utilizand XML Parser-ul JDom. Aceasta este implementata in pachetul `ro.dazoot.maze3d.config` si de ea se ocupa clasa `ReadXMLFile.java`.

Nota:

JDom se poate downloada de la www.jdom.org

Pentru a parsea fisierul .xml de definire a labirintului se folosesc functiile continute in clasa `SAXBuilder` oferita de JDom. Functia urmatoare are ca scop obtinerea documentului xml:

```
protected Document getXMLDocument() throws ConfigException
```

```

{
    SAXBuilder builder = new SAXBuilder();
    builder.setIgnoringElementContentWhitespace(true);

    try
    {
        Document doc;

        switch(this.mode)
        {
            case URL_MODE:
            {
                doc = builder.build(url);
                break;
            }
            case STREAM_MODE:
            {
                doc = builder.build(stream);
                break;
            }
            case FILENAME_MODE:
            {
                doc = builder.build
                    (new File(filename));
                break;
            }
            default:
            {
                throw new ConfigException
                    ("No such mode: "+this.mode);
            }
        }

        return doc;
    }
    catch (JDOMException jde)
    {
        System.err.println(jde.getMessage());
        throw new ConfigException("Nu am putut
            citi fisierul xml.");
    }
}

```

Pentru a obtine datele pentru fiecare element care ne intereseaza am construit in cadrul clasei ReadXMLFile functiile readWall() care returneaza un obiect de tip Wall si readCell() care returneaza un obiect de tip Cell. In randurile urmatoare este prezentata functia de citire a unui perete, cea pentru celula fiind similara.

```

protected Wall readWall(Element node)
{
    Wall wall=new Wall();

```

```
String texture_id, type, normaltype;

Map attribs=getAttribsMap(node);
texture_id=(String)attribs.get("texture");
type=(String)attribs.get("type");
normaltype=(String)attribs.get("normaltype");

if (texture_id==null)
{
    attribs.put("texture",
        conf.getDefault_cell().getWall(type).getTexture());
}
else attribs.put("texture", conf.getTexture(texture_id));

if (normaltype==null)
{
    attribs.put("normaltype",
        "+conf.getDefault_cell().getWall(type).
        getNormaltype());
}

try
{
    BeanUtils.populate(wall, attribs);
} catch (Exception e)
{
    // nothing here we add it blank
    e.printStackTrace();
}
return wall;
}
```

2.2. Construirea labirintului

2.2.1. Elemente virtuale

O data obtinute datele din fisier acestea trebuiesc retinute in diverse obiecte si afisate in format grafic pe ecran. Obiectele virtuale care retin caracteristicile si elementele obtinute prin parsarea fisierului .xml sunt implementate in pachetul `ro.dazoot.maze3d.elements`. Elementele "virtuale" definite pentru aceasta aplicatie sunt: celula (`Cell.java`), perete (`Wall.java`) si textura (`Texture.java`). Acestea retin doar caracteristicile fizice ale obiectelor desemnate, inasa nu cunosc nimic despre partea de desenare propriu-zisa ale acestora, fiind de fapt `JavaBeans` ce sunt populate la citirea fisierului .xml.

Spre deosebire de `Wall` si `Cell` care contin setteri si getteri, clasa ce defineste un obiect de tip textura are in plus o functie care incarca imaginea corespunzatoare texturii respective.

```
private void loadTexture() throws Exception
{
    System.err.println("Loading texture: "+filename);
    try {
        texture2d = (Texture2D)TextureLoader.tf.getMinMapTexture(filename);
        //bt=TextureLoader.tf.getBufferedImageAlpha(filename);
    } catch (Exception ex)
    {
        ex.printStackTrace();
        throw new Exception("Could not load texture "+filename);
    }
}
}
```

2.2.2. Elemente grafice

Corespunzatoare peretelui si celulei "virtuale", in pachetul `ro.dazoot.maze3d.elements3d` se gasesc clasele ce definesc grafica obiectului 3D(`Cell3D.java` si `Wall3D.java`). Acestea au rolul de a construi elementele grafice ce vor fi afisate in joc.

Astfel, un perete este un obiect de tip `Shape3D`, doua triunghiuri ce formeaza un patrat, caruia i se calculeaza punctele de definire stiind coltul stanga jos al celulei, dimensiunea sa si tipul de perete. In fragmentul de cod urmator este prezentat modul de atribuire a celor 6 punctelor ce formeaza cele doua triunghiuri.

```
if (type.equals("podea"))
{
    coords=new Point3f[] {
        new Point3f(x*w, y*h, z*w),
        new Point3f((x+1)*w, y*h, z*w),
        new Point3f((x+1)*w, y*h, (z+1)*w),
        new Point3f((x+1)*w, y*h, (z+1)*w),
        new Point3f(x*w, y*h, (z+1)*w),
        new Point3f(x*w, y*h, z*w),
    };
} else if (type.equals("tavan"))
{
    coords=new Point3f[] {
        new Point3f(x*w, (y+1)*h, z*w),
        new Point3f((x+1)*w, (y+1)*h, z*w),
        new Point3f((x+1)*w, (y+1)*h, (z+1)*w),
        new Point3f((x+1)*w, (y+1)*h, (z+1)*w),
        new Point3f(x*w, (y+1)*h, (z+1)*w),
        new Point3f(x*w, (y+1)*h, z*w),
    };
} else if (type.equals("stanga"))
{
    coords=new Point3f[] {
```

Maze3D - Remote Mobile Control

```
        new Point3f(x*w, y*h, z*w),
        new Point3f(x*w, y*h, (z+1)*w),
        new Point3f(x*w, (y+1)*h, (z+1)*w),
        new Point3f(x*w, (y+1)*h, (z+1)*w),
        new Point3f(x*w, (y+1)*h, z*w),
        new Point3f(x*w, y*h, z*w),
    };
} else if (type.equals("dreapta"))
{
    coords=new Point3f[] {
        new Point3f((x+1)*w, y*h, z*w),
        new Point3f((x+1)*w, y*h, (z+1)*w),
        new Point3f((x+1)*w, (y+1)*h, (z+1)*w),
        new Point3f((x+1)*w, (y+1)*h, (z+1)*w),
        new Point3f((x+1)*w, (y+1)*h, z*w),
        new Point3f((x+1)*w, y*h, z*w)
    };
} else if (type.equals("fata"))
{
    coords=new Point3f[] {
        new Point3f(x*w, y*h, (z+1)*w),
        new Point3f((x+1)*w, y*h, (z+1)*w),
        new Point3f((x+1)*w, (y+1)*h, (z+1)*w),
        new Point3f((x+1)*w, (y+1)*h, (z+1)*w),
        new Point3f(x*w, (y+1)*h, (z+1)*w),
        new Point3f(x*w, y*h, (z+1)*w),
    };
} else // spate
{
    coords=new Point3f[] {
        new Point3f(x*w, y*h, z*w),
        new Point3f((x+1)*w, y*h, z*w),
        new Point3f((x+1)*w, (y+1)*h, z*w),
        new Point3f((x+1)*w, (y+1)*h, z*w),
        new Point3f(x*w, (y+1)*h, z*w),
        new Point3f(x*w, y*h, z*w),
    };
}
```

Acestui Shape3D obtinut i se adauga textura (si coordonatele sale), normala si i se seteaza transparenta daca este cazul. Am ales sa nu desenam un perete ca un patrat, ci ca un triunghi pentru a fi mai usor de calculat normala ce serveste la detectia coliziunilor. Coordonatele pentru textura ce se aplica sunt alese astfel incat textura sa se intinda pe toata dimensiunea peretelui.

```
TexCoord2f[] texCoords = new TexCoord2f[] {
    new TexCoord2f(0f,0f),
    new TexCoord2f(1f,0f),
    new TexCoord2f(1f,1f),
    new TexCoord2f(1f,1f),
}
```

```
        new TexCoord2f(0f,1f),  
        new TexCoord2f(0f,0f),  
};
```

O celula este o reuniune de (maxim) 6 pereti: podea, tava, spate, fata, stanga, dreapta. Se deseneaza doar peretii care au proprietatea de a fi prezenti in celula respectiva. In cadrul celulei se seteaza, pentru fiecare perete existent si suprafata de coliziune determinata de aceasta (pentru fiecare perete existent), mai putin tavanul.

In cadrul unei celule pot fi adaugate diverse elemente, obiecte grafice, daca acestea sunt specificate ca `extrarenderer` in fisierul `world.xml` la definirea celulei respective. Aceste "plugin-uri" pentru celule sunt definite in pachetul `ro.dazoot.maze3d.renderplugins` si implementeaza toate interfaata definita in `RenderPlugin.java`.

```
public interface RenderPlugin  
{  
    public Node draw(Cell cell);  
    public List getColliders();  
}
```

In momentul de fata am implementat 3 "plugin-uri" diferite ce se pot gasi in interiorul unei celule. Acestea sunt: `Sphere.java`, `Thorus.java` si `Water.java`.

2.2.3. Miscarea prin labirint

Miscarea prin labirint se realizeaza prin aplicarea unei forte de tip `MovementForce` implements `Force`. Aceasta forta este un obiect care are setate functii pentru fiecare tip de miscare, si dupa efectuarea uneia updateaza view-ul. Miscarea este realizata cu o anumita viteza ce poate fi setata. La declansarea impulsului de o anumita miscare (prin sageti de la tastatura sau prin intermediul mobilului) `MovementForce` seteaza valoarea actiunii urmatoare egala cu o constata predefinita. De exemplu pentru intoarcerea la stanga avem:

```
public void turnLeft()  
{  
    turnDirection = TURN_LEFT;  
}
```

O data setata valoarea actiunii urmatoare, in cadrul functiei `updateView(..)` se incrementeaza corespunzator pozitia in cadrul labirintului.

```
switch (turnDirection)  
{
```

```
case TURN_STOP:
    turnAngle.stop();
    break;
case TURN_RIGHT:
    turnAngle.startDecreasing(totalTime);
    break;
case TURN_LEFT:
    turnAngle.startIncreasing(totalTime);
    break;
}
```

2.2.4. Detectia coliziunilor

Problema detectiei coliziunilor a aparut din necesitatea de a nu se permite trecerea prin peretii labirintului sau prin diverse obiecte aflate in interior. Rezolvarea este simpla: cel care se plimba prin labirint este vazut ca o sfera si se calculeaza de fiecare data posibila coliziune cu obiectele din jur. Acest lucru se realizeaza prin intermediul obiectelor de tip `ColliderNode`. Utilizatorul, cel care se plimba este definit astfel:

```
private ColliderNode avatarNode;
avatarNode = new ColliderNode(n,
ColliderNode.CT_BOUNDING_SPHERE,
ColliderNode.CT_BOUNDING_SPHERE,
true,
new Coord3f(1f, 1, 1f));
```

Acestui `avatarNode` i se adauga o forta de miscare prin care se efectueaza deplasarea.

Pentru a se defini elementele cu care `avatarNode` ar putea intra in coliziune s-a construit un `CollisionSystem`, caruia i s-a adaugat cate un `ColliderNode` corespunzator fiecarui obiect aflat in scena (fie el perete sau "plugin" al celulei). La aparitia unei coliziuni intre `avatarNode` si un obiect se executa o actiune predefinita. Se putea alege din mai multe actiuni, si anume: `STOP`, `SLIDE` sau `BOUNCE`. Am considerat ca cea mai realista este aceea de `SLIDE`, si anume la contactul cu un perete, daca se mentine forta ce conduce inspre perete sa se efectueze o alunecare de-a lungul acestuia.

`ColliderNode`-urile corespunzatoare fiecarui perete sunt setate la crearea fiecarei celule grafice. `ColliderNode`-ul pentru un plugin este setat in cadrul clasei ce mplementeaza respectivul obiect. Fiecare celula are o lista de collideri, ce vor fi adaugati intr-un final la `CollisionSystem`-ul labirintului.

Setarea colliderilor pentru fiecare dintre peretii unei celule:

```
// setam collider (pentru collision test)
ColliderGeometry cg = new ColliderGeometry();
cg.setModel(w3d);
```

```
BiTreeCollider bic = new BiTreeCollider();
bic.build(cg);
ColliderNode cn =
    new ColliderNode(
        w3d, ColliderNode.CT_GEOMETRY,
        ColliderNode.CT_GEOMETRY,
        false,
        bic);
cn.setTwoSided(true);
cn.setResponse(CollisionResponse.BOUNCE);
this.colliders.add(cn);
```

Adaugarea listei de collideri corespunzatori unei celule:

```
cs = new CollisionSystem(-1);

Iterator it=controller.getConfiguration().getCells().iterator();
while(it.hasNext())
{
    c=(Cell)it.next();
    c3d=new Cell3D(c);
    col_it=c3d.getColliders().iterator();
    while(col_it.hasNext())
    {
        cs.addCollider((ColliderNode)col_it.next());
    }
    scene.addChild(c3d);
}
```

Avatarul este deasemenea adaugat la CollisionSystem, el fiind o BILA (sfera) si are proprietatea de a fi mobile (MobileCollider).

2.3. Maze3D SERVER

Pentru a face posibila partea de Remote Control, am implementat in cadrul aplicatiei un mic SERVER care poate dialoga cu doua(2) tipuri de clienti.

Client de tipul 1 - Mobile Client

si Client de tipul 2 - Computer Client.

Partea de server se gaseste in `ro.dazoot.maze3d.net.*` package.

Serverul porneste implicit pe portul 1000 si asteapta conexiuni de la clienti (Socket).

Protocolul de comunicatie este relativ simplu, si anume:

- Clientul se conecteaza la server si trimite un string de forma: HELLO [tip] [id] (exemplu: HELLO 2 spg)
- Serverul raspunde cu mesaj de greeting RESPT [string here]
- Clientul trimite comenzi de genul ACTION [params of action here] (exemplu: ACTION FORWARD START sau ACTION FORWARD STOP)

Maze3D - Remote Mobile Control

- Serverul primește comanda ACTION și caută în lista de clienți conectați corespondentul clientului curent, adică dacă eu sunt 1 spg, el caută 2 spg și invers, după care trimite comanda către clientul corespondent.

De remarcat că în cazul conectării de pe mobil, datorită faptului că MIDP1.0 (Nokia 7250, Series 40) nu suportă decât `URLConnection` partea de comunicare are loc simulând un server WEB.

Drept urmare comenzile de la telefonul mobil nu sunt continue, conectarea făcându-se la fiecare comandă trimisă.

Clientul Mobil (Telefonul) trimite un REQUEST HTTP GET având în headerul requestului o cheie de forma:

```
DATA: [tip] [id] ACTION [params of action here] (exemplu: DATA: 1 spg ACTION BACKWARD STOP).
```

Serverul implementat detectează dacă avem de-a face cu un client HTTP (mobilul) și modifică răspunsul în așa fel încât să simuleze un server de web.

2.4. Maze3D Clients

2.4.1. Computer Client

Clientul COMPUTER este atașat părții grafice și nu face decât să se conecteze la server într-un thread separat, să se identifice ca fiind de tipul 2 și să aștepte comenzi de la un client corespondent, adică de tipul 1 (Mobile Client).

Recunoaște comenzi de genul ACTION [params here] pe care le aplică forței de mișcare `MovementForce`.

Nota:

Datorită delay-ului conexiunii HTTP de pe clientul mobil, la `turnLeft` și `turnRight`, acțiunea este oprită după un număr de milisecunde predefinit, deoarece în lipsa acestei acțiuni întârziate (de oprire) se produce efectul învârtirii în jurul "cozii", lucru care nu este deloc plăcut (amator).

2.4.2. Mobile Client

Clientul MOBILE (telefonul mobil, în cazul nostru un amarat de Nokia 7250) tratează `keyPress` și `keyReleased` și în funcție de tasta apăsată trimite comanda la server.

Bineînțeles acest lucru se face trimițând datele de identificare cu fiecare REQUEST.

Nota:

Pentru a evita cazurile în care tastele sunt apăsate prea REPEDE, trimiterea de mesaje este sincronizată, folosind o variabilă booleană de control.

Exemplu de functionare:

Utilizatorul apasa pe 2 (up) - Mobilul trimite print-o conexiunea HTTP la adresa presetata in sursa, o comanda de genul ACTION FORWARD START.

Utilizatorul elibereaza tasta 2(up) - Mobilul trimite o comanda de genul ACTION FORWARD STOP

In cazul LEFT, RIGHT doar comanda de START este trimisa deoarece ea este executata cu un timer de cateva milisecunde, dupa care este intrerupta. (vezi mai sus).

3. Linkuri

- Xith3D <http://www.xith3d.org>
- Java <http://java.sun.com>
- OpenGL <http://www.opengl.org>
- JOGL <http://jogl.dev.java.net>
- Eclipse <http://www.eclipse.org>
- Laborator SPG <http://spg.ss.pub.ro> (Sisteme de Prelucrare Grafica)

4. Credits

Nume	Email
Indrumator stiintific: Octavian Georgescu	octavian.at.ss.pub.ro
Programator: Daniela Tiba	dana.at.dazoot.ro
Programator: Constantin Catalin	catalin.at.dazoot.ro