

This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. The article is believed to be still valid for the original topic scope.



Contents

- [1 About this widget](#)
- [2 WidSets Scripting Language code: MemoryGame.he](#)
- [3 Widget.xml](#)
- [4 See also](#)

About this widget

This is a memory game widget.

WidSets Scripting Language code: MemoryGame.he

```
class MemoryGame
{
    const int CMD_BACK      = 1;
    const int CMD_RESET    = 2;
    const int CMD_OPEN     = 4;
    const int CMD_SAVE     = 6;
    //const int DEBUG      = 6;

    const String VIEW_MAXI = "maxiView";
    const String VIEW_HIGN = "highScores";

    struct Cell {
        boolean opened;
        boolean found;
        int fruit;
    }

    MenuItem OPTIONS = new MenuItem(OPEN_MENU, "Options");
    MenuItem BACK    = new MenuItem(CMD_BACK, "Back");

    Menu MENU = new Menu()
        .add(CMD_RESET, "Reset");

    int      ready = 0;
    boolean  revealing = false;
    long     elapsed = 0;
    long     m_started = 0;
    boolean  finished = false;
    boolean  wrongFruit;
    Shell    maxiView;
    Flow     m_flow;
    Flow     t_flow;
    Flow     s_flow;
```

Memory_Game

```
Label      m_timelabel;
Timer      m_timer;
Image      back = getImage("back100.png");
Picture    pic;
Picture    pic2;
List       fruitLottery = new List();
List       fruits = new List();
String     httpService = getServices()["http"];
Input      nickInput;
Shell      highScore;
boolean    hs = false;

Component createElement(String viewName, String elementName, Style style, Object context) {
    if (elementName.equals("maxi")) {
        m_flow = new Flow(getStyle("mini"));
        m_flow.setPreferredSize(-100, -100);
        return new Scrollable(style, m_flow);
    } else if (elementName.equals("time")) {
        t_flow = new Flow(getStyle("time"));
        t_flow.setPreferredSize(-100, -100);
        m_timelabel = new Label(getStyle("time"), "");
        m_timelabel.setPreferredSize(-100, -100);
        t_flow.add(m_timelabel);
        return t_flow;
    } else {
        return null;
    }
}

void startWidget() {
    setMinimizedView(createMinimizedView("miniView", null));
}

Shell openWidget() {
    fruits
        .add(getImage("front1.png"))
        .add(getImage("front2.png"))
        .add(getImage("front3.png"))
        .add(getImage("front4.png"))
        .add(getImage("front5.png"))
        .add(getImage("front6.png"))
        .add(getImage("front7.png"))
        .add(getImage("front8.png"))
        .add(getImage("front9.png"))
        .add(getImage("front10.png"))
        .add(getImage("front11.png"))
        .add(getImage("front12.png"))
        .add(getImage("front13.png"))
        .add(getImage("front14.png"))
        .add(getImage("front15.png"));
    maxiView = new Shell(createMaximizedView("maxiView", null));
    m_timer = schedule(1000, 1000);
    finished = false;
    hs = false;
    ready = 0;
    lottery();
    createBoard();
    return maxiView;
}

void updateTime() {
```

Memory_Game

```
elapsed = (currentTimeMillis() - m_started) / 1000;
m_timelabel.setText(String(elapsed));
m_timelabel.repaint(false);
flushScreen(false);
}

void lottery() {
    for(int k=0; k<2; k++) {
        for(int i=0; i<15; i++) {
            int j = (i);
            fruitLottery.add(String(j));
        }
    }
}

Cell getCell(int key) {
    return Cell(m_flow[key].getData());
}

void revealFruit(Picture current) {
    revealing = true;
    if (current != pic && !Cell(current.getData()).found && !Cell(current.getData()).opened) {
        Cell cell2 = null;
        if (pic != null) {
            pic2 = pic;
            cell2 = Cell(pic2.getData());
        }
        pic = current;
        Cell cell = Cell(pic.getData());

        if (!cell.found) {
            if(!cell.opened) {
                pic.setImage(Image(fruits[cell.fruit]));
                cell.opened = true;
                pic.repaint(false);
            }
            if (cell2 != null) {
                if(cell.opened && cell2.opened) {
                    if (cell.fruit == cell2.fruit) {
                        ready++;
                        cell.found = true;
                        cell.opened = false;
                        cell2.found = true;
                        cell2.opened = false;
                        pic.setImage(Image(fruits[cell.fruit]));
                        pic2.setImage(Image(fruits[cell2.fruit]));
                        pic.repaint(false);
                        pic.repaint(false);
                    } else if (cell2.opened && !cell.found && !cell2.found) {
                        wrongFruit = true;
                        cell.opened = false;
                        cell2.opened = false;
                    }
                }
            }
        }
        if (ready == 15) {
            finished = true;
            m_timer.cancel();
            done(elapsed, true);
        }
    }
}
```

Memory_Game

```
    revealing = false;
}

void createBoard() {
    hs = false;
    finished = false;
    m_flow.clear();
    m_timer = schedule(1000, 1000);
    m_started = currentTimeMillis();
    for(int i=0; i<6; i++) {
        for(int j=0; j<5; j++) {
            Cell cell = new Cell();
            pic = new Picture(getStyle("cell"), back);
            pic.setFlags(j == 4? VISIBLE|FOCUSABLE|LINEFEED : VISIBLE|FOCUSABLE);
            int fruitTemp = random(fruitLottery.size());
            cell.fruit = int(String(fruitLottery[fruitTemp]));
            fruitLottery.remove(fruitTemp);
            pic.setData(cell);
            pic.setPreferredSize(-100/5, -100/6);
            pic.setAction(CMD_OPEN);
            m_flow.add(pic);
        }
    }
    m_flow.repaint(false);
}

void done(long time, boolean ok) {
    printf("done funkkari: "+getUsername());
    m_flow.clear();
    s_flow = new Flow(getStyle("maxi"));
    Picture congratule = new Picture(getStyle("congrat"), getImage("congrat"));
    congratule.setPreferredSize(-100,-20);
    congratule.setFlags(VISIBLE|LINEFEED);
    s_flow.add(congratule);
    Label elapsed = new Label(getStyle("text"), "Your score: "+time+" sec");
    elapsed.setPreferredSize(-100,-15);
    elapsed.setFlags(VISIBLE|LINEFEED);
    s_flow.add(elapsed);
    Label resetb = new Label(getStyle("button"), "Play Again");
    resetb.setAction(CMD_RESET);
    resetb.setPreferredSize(-100,-15);
    resetb.setFlags(VISIBLE|FOCUSABLE|LINEFEED);
    s_flow.add(resetb);
    Label empty = new Label(getStyle("scoreboard"), "");
    empty.setPreferredSize(-100,-35);
    s_flow.add(empty);
    highScore = new Shell(s_flow);
    pushShell(highScore);
}

void createScores(String data) {
    printf("scores" + data);
    int offset = 0;
    int last = 0;
    Picture scores = new Picture(getStyle("congrat"), getImage("highscores"));
    scores.setFlags(VISIBLE|LINEFEED);
    scores.setPreferredSize(-100,-20);
    s_flow.add(scores);
    int pos = 1;
    int fh = 20;
    while(last != -1) {
        last = data.indexOf('|', offset);
```

Memory_Game

```
if (last != -1) {
    scores.setAction(CMD_OPEN);
    String line = data.substring(offset, last);
    int point = line.indexOf(',', 0);
    Label position = new Label(getStyle("scoreboard"), pos+".");
    Label nick = new Label(getStyle("scoreboard2"), line.substring(0, point));
    Label score = new Label(getStyle("scoreboard"),line.substring(point+1, line.length()));
    position.setFlags(VISIBLE);
    nick.setFlags(VISIBLE);
    score.setFlags(VISIBLE|LINEFEED);
    position.setPreferredSize(-20, -10);
    nick.setPreferredSize(-40, -10);
    score.setPreferredSize(-40, -10);
    s_flow.add(position);
    s_flow.add(nick);
    s_flow.add(score);
    fh=fh+10;
    pos++;
}
offset = last+1;
}
if (fh < 100) {
    Label empty = new Label(getStyle("scoreboard"), "");
    empty.setPreferredSize(-100, -(100-fh));
    s_flow.add(empty);
}
s_flow.repaint(false);
flushScreen(true);
}

void reset(String set) {
    m_flow.clear();
    lottery();
    createBoard();
}

Menu getMenu(Shell shell, Component focused)
{
    return MENU.reset();
}

boolean keyAction(Component source, int op, int code) {
    if (op == KEY_PRESSED) {
        if (wrongFruit) {
            wrongFruit = false;
            pic.setImage(back);
            pic.repaint(false);
            pic2.setImage(back);
            pic2.repaint(false);
            pic = null;
            return false;
        }
    }
    return false;
}

MenuItem getSoftKey(Shell shell, Component focused, int key) {
    if (key == SOFTKEY_OK) {
        if (hs) {
            return null;
        } else {
            return OPTIONS;
        }
    }
}
```

Memory_Game

```
    }
} else if (key == SOFTKEY_BACK) {
    return BACK;
}
return null;
}

void actionPerformed(Shell shell, Component source, int action) {
    switch(action)
    {
    case CMD_BACK:
        {
            if (finished && hs) {
                hs = false;
                popShell(shell);
            } else if (hs) {
                hs = false;
                m_flow.clear();
                flushScreen(true);
                ready = 0;
                elapsed = 0;
                lottery();
                createBoard();
                popShell(shell);
            } else if (finished) {
                finished = false;
                m_flow.clear();
                flushScreen(true);
                hs = false;
                ready = 0;
                elapsed = 0;
                lottery();
                createBoard();
            }
            m_timer.cancel();
            //fruits = new List();
            popShell(shell);
        }
        break;

    case CMD_SAVE:
        {
            String nick = nickInput.getText();
            if (nick.length() > 1 && nick.length() < 11) {
                popShell(shell);
            } else {
                done(elapsed, false);
            }
        }
        break;

    case CMD_RESET:
        {
            if (hs) {
                popShell(shell);
                hs = false;
            } else if (finished) {
                finished = false;
                popShell(shell);
            }
            m_flow.clear();
            flushScreen(true);
        }
    }
}
```

Memory_Game

```
        ready = 0;
        elapsed = 0;
        lottery();
        createBoard();
    }
    break;

case CMD_OPEN:
    {
        if (!wrongFruit && !revealing) {
            revealFruit(Picture(source));
        }
    }
    break;

}

}

void timerEvent(Timer timer)
{
    if (!finished) {
        updateTime();
    }
}
}
```

Widget.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<widget spec_version="2.0">

    <info>
        <name>MemoryGame</name>
        <version>1.0</version>
        <author>Pekka</author>
        <shortdescription>Fruitsets Memory Game</shortdescription>
        <longdescription>Find the fruitpairs as fast as you can
            to get your name on the highscore board!</longdescription>
        <tags>games fun</tags>
    </info>

    <services>
        <service type="http" id="send"/>
        <service type="http" id="get"/>
    </services>

    <resources>
        <stylesheet>
            button
            {
                align: hcenter vcenter;
                font-1: small bold;
                color-1: blue;
                background: grid9 "button.png" 10 10 10 10;
                margin: 2 30 2 30;

                focused
                {
                    align: hcenter vcenter;
```

Memory_Game

```
        font-1: small bold;
        color-1: red;
        background: grid9 "button2.png" 10 10 10 10;
    }
}

congrat
{
    align: hcenter vcenter;
    font-1: medium bold;
    color-1: red;
    background: vgradient #b2e8fe white;
}

text
{
    align: hcenter vcenter;
    font-1: small bold;
    color-1: black;
}

scoreboard
{
    align: hcenter vcenter;
    font-1: small bold;
    color-1: red;
    background: solid white;
}

scoreboard2
{
    align: hcenter vcenter;
    font-1: small bold;
    color-1: blue;
    background: solid white;
}

bkg
{
    align: hcenter vcenter;
    background: grid9 "bkg.png" 6 6 6 6;
}

time
{
    align: hcenter top;
    font-1: large bold;
    color-1: black;
}

minbkg
{
    align: hcenter vcenter;
    background: grid9 "bkg.png" 20 20 20 20;
}

mini
{
    align: hcenter top;
}
```

Memory_Game

```
maxi
{
    align: hcenter vcenter;
    background: solid white;
}

maxiView
{
    align: hcenter top;
    background: grid9 "bkg.png" 20 20 20 20;
}

cell
{
    align: hcenter vcenter;
    margin: 2 2 2 2;

    focused
    {
        align: hcenter vcenter;
        border-type: rectangle red;
        border: 2 2 2 2;
        margin: 0 0 0 0;
    }
}
</stylesheet>






















<code src="MemoryGame.he"/>
</resources>

<parameters>
    <parameter name="widgetname" type="string" description="Name of widget" help="" editable="false"
        protected="false" visible="true" sendmobile="true">
        MemoryGame
    </parameter>
</parameters>

<layout minimizedheight="45sp">
    <view id="highScores" class="scoreboard" top="0%" right="100%" bottom="100%" left="0%">
</view>
```

Memory_Game

```
<view id="miniView" top="0%" right="100%" bottom="100%" left="0%">
  
</view>

<view id="maxiView" class="maxiView" top="0%" right="100%" bottom="100%" left="0%">
  <script class="maxi" id="maxi" top="10%" right="90%" bottom="90%" left="10%"/>
  <script class="time" id="time" top="90%"/>
</view>
</layout>

</widget>
```

See also

- [WidSets SDK](#)
- [WidSets Client](#)
- [WidSets Scripting Language](#)
- [Widget examples](#)
 - ◆ [WidClock](#)
 - ◆ **Memory Game**
 - ◆ [Filter test](#)
 - ◆ [Hello World](#)
 - ◆ [UITest](#)