JSGLib Tutorial 2: Dice

Goal of this tutorial is to roll some dice and demonstrate how to change the appearance of stage objects and handle key input. It could easily be turned into a game but as the focus is on how to use the *jsglib* we don't go that far.

Dice.java

Therefore, we first create *Dice.java* in package *dice*:

```
package dice;
import jsglib.*;

public class Dice {
     StageObject obj;
     int spots;
     public Dice(Stage s, int x, int y) {
          obj = new StageObject(s,x,y,"dice1.png",40);
          obj.hide();
          spots = 1;
          obj.addImage("dice2.png");
          obj.addImage("dice3.png");
          obj.addImage("dice4.png");
          obj.addImage("dice5.png");
          obj.addImage("dice6.png");
     }
}
```

hide hides the object from the stage, can be shown again by *show*.

addImage adds another image to the object. We specify no id for the images (we could use *addImage(String id, String fname)* for this). If no id is given the file name is the id. If the stage object gets an id itself via the constructor the first image uses this id. Either you can later on set an image by using the id or the index in the images list starting with index 0. We use the later as it is more convenient for our example.

Program.java

In *Program.java* we create 5 dice:

```
package dice;
import jsglib.*;

public class Program {
     public static void main(String[] args) {
          Stage stage = new Stage("Dice",800,600,"rock.jpg");
          Dice dice[] = new Dice[5];

          for (int i = 0; i < 5; i++)
               dice[i] = new Dice(stage,100 + i*150,300);
     }
```

```
}
```

Roll the Dice

We don't see anything yet as we hid the dice. Now add the *roll* method to class *Dice*:

```
public void roll() {
     obj.show();
     for (int i = 0; i < 360; i += 20) {
          spots = Tools.rand(1,6);
          obj.setImage(spots-1);
          obj.rotate(20);
          Tools.wait(20);
     }
}
```

We first show the dice again and „roll“ it with a small animation and remember the current spots. Note, that *rotate* goes clockwise with positive values. You could also use *rotateRight(deg)* or *rotateLeft(deg)* to avoid confusion. *setImage(int)* sets the image by index. As we added the images in increasing order *dice1.png* is at index 0, *dice2.png* at index 1, … so we can easily use *spots – 1* as index. If, e.g., you had added *dice2.png* as *addImage("d2", "dice2.png");* you could have set the image by *setImage("d2");*.

Now roll the dice in main after waiting for a key to start:

```
Tools.println("Press any key to roll dice.");
stage.waitForKey();

for (int i = 0; i < 5; i++)
     dice[i].roll();
```

waitForKey waits for any key to be pressed and then released. It also returns the pressed key as String (if any letter from A to Z, digit from 0 to 9 or SPACE, ENTER, UP, DOWN, LEFT, RIGHT).

Note, that the dice will be rolled one after the other. As we use a single thread and wait in class *Dice*, the dice can't be rolled simultaneously. Either you'd use a non-waiting update method with a single wait in the main loop or multiple threads (the latter being the better solution and subject in the next tutorial).

Hold Dice

Next, add *boolean hold* and method *hold* to *Dice.java* to „freeze“ a dice so that *roll* does not re-roll it if on hold (just add the instructions written in bold):

```
public class Dice {
     ...
     boolean hold;

     public Dice(Stage s, int x, int y) {
```

```
            ...
            hold = false;
      }

      public void roll() {
            if (hold)
                  return;
            ...
      }

      public void toggle() {
            hold = !hold;
            if (hold)
                  obj.moveTo(obj.getX(), obj.getY()+100);
            else
                  obj.moveTo(obj.getX(), obj.getY()-100);
      }

}
```

To visualize its state a dice on hold is moved down a little. *moveTo* either takes a position x,y or another object. *getX* and *getY* get an object's center position.

Now add the following written in bold to *main*:

```
      ...
      for (int i = 0; i < 5; i++)
            dice[i].roll();

      while (true) {
            if (stage.isKeyReleased("A"))
                  dice[0].toggle();
            if (stage.isKeyReleased("S"))
                  dice[1].toggle();
            if (stage.isKeyReleased("D"))
                  dice[2].toggle();
            if (stage.isKeyReleased("F"))
                  dice[3].toggle();
            if (stage.isKeyReleased("G"))
                  dice[4].toggle();
            if (stage.isKeyReleased("SPACE"))
                  for (int i = 0; i < 5; i++)
                        dice[i].roll();
            Tools.wait(20);
      }
```

isKeyReleased returns true if key was pressed and then released. You must not use *isKeyPressed* here as it returns true as long as the key is pressed resulting in the dice to jump around. That a key was released is remembered for 10 program cycles (200 ms by default) so your delay shouldn't be too high otherwise the release event is lost (again just

stick to 20 ms, it's really a reasonable value).

As we work around real event handling here to make it easier for students it is a bit unclean: As mentioned the delay must not be too high and if used in multiple threads each key must only be checked in one thread otherwise events get lost again. On the other hand it doesn't make too much sense to check the same key for different objects. So both limitations are ok in my opinion.