Turtle-World Programmer's Manual

Memorial University 2003 — 2012

Dennis Peters & Theodore Norvell

What is Turtle-World

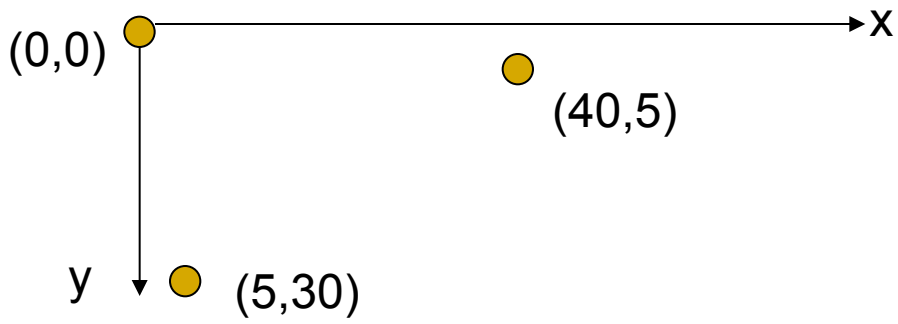
Turtle-world is intended to make graphics programming in Java a bit more accessible and fun.

- Shapes can be drawn on the screen by sending commands to a simulated robotic turtle.

Controlling the turtle

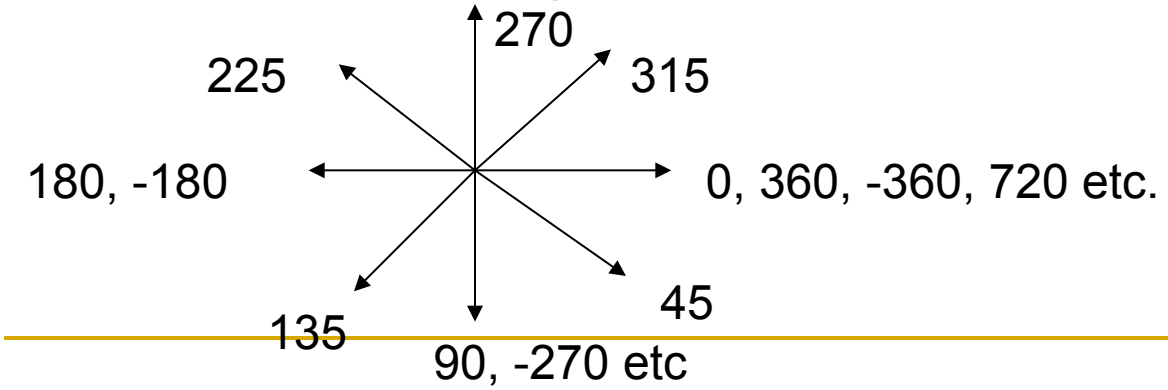
The turtle's position, velocity, and acceleration are controlled with the following commands

- crush.setPosition(x, y) ;
  - x and y are rectangular coordinates as follows.
  - the units are pixels (the smallest rectangle that the screen can show).



Controlling the turtle (cont.)

- crush.setSpeed(v) ;
  - Set how fast the turtle is travelling
  - Units are pixels per second
- crush.setOrientation(alpha)
  - Set the direction of motion
  - Units are degrees clockwise from the x-axis



Controlling the turtle (cont.)

- crush.setVelocity(vx, vy) ;
  - An alternative way of setting velocity.
  - Units are pixels / second

Controlling the turtle (cont.)

- crush.setRateOfChangeOfSpeed(a) ;
  - Controls the rate of change of the speed
  - Units are pixels / second / second
- crush.setSpin(omega) ;
  - Controls the rate of change of orientation
  - Units are degrees / second
- crush.reset() ;
  - Sets all position, speed, rate of change of speed and spin attributes to 0.0.

Interrogating the Turtle

- To find out about the Turtle's state, we can use methods that return values.
- **double** x = crush.getPositionX() ;
- **double** y = crush.getPositionY() ;
- **double** s = crush.getSpeed() ;
- **double** alpha = crush.getOrientation() ;
- **double** vx = crush.getVelocityX() ;
- **double** vy = crush.getVelocityY() ;

Interrogating the Turtle (cont.)

- **double** a = crush.getRateOfChangeOfSpeed() ;
- **double** omega = crush.getSpin() ;

Making your own buttons

- You can add buttons to the Turtle-talk application by:
  - Creating a new “method” in the TurtleController class. This method should be “public”, “void”, and have no parameters.
  - Adding the name of the method to a list called “buttons” in the TurtleController class.

Making your own buttons (cont)

Example

- In the editor. Add the following lines to the TurtleController class

```
public void circle() {
    crush.setOrientation(0.0) ;
    crush.setAngularVelocity( 45.0 );
    crush.setSpeed( 50.0 ) ;   }
```

- Find the “buttons” list near the top of the TurtleController class. Add “circle” to the list.

Making your own buttons (cont)

- Select “Run” from the “Run” menu.
- Note the “circle” button. What happens if you click on it?

Making your own buttons (cont)

- What's happening?
  - When you click on the "circle" button, a "circle()" message is sent to a "TurtleController" object where the message causes execution of the "circle" method.
  - The "circle" method of the "TurtleController" object sends three messages to the object named "crush". These messages change the initial orientation, the speed, and the angular velocity of the turtle object.
  - A graphical representation of the turtle object is drawn on the screen.

Adding Check-Boxes

- Check-boxes can be checked and unchecked by the user.
- To add a check-box you must do two things
  - Add the name of the check-box to the list named “checkBoxes” at the top of the TurtleController class.
  - Add a two methods to the TurtleController that are public and void and have no parameters. The names of these methods should be *nameOn* and *nameOn* where *name* is the name of the check-box.

Adding Check-boxes example

- Consider adding a check-box called “swerve”
- Add “swerve” to the checkBox list

```
public static String[] checkBoxes = new String[] { "wiggle",
    "checkIntersections", “swerve" } ;
```

- Messages “swerveOn” and “swerveOff” will be sent to the TurtleController when the box is checked or unchecked.
- Add a **boolean** name “swerve” to the class outside of any method

```
private boolean swerve = false ;
```

- Add a “swerveOff” method

```
public void swerveOff() {swerve = false ; }
```

Adding Check-boxes example

- Add a swerveOn method

```
public void swerveOn() {
    swerve = true ;
    crush.setRadarRange(50.0);
    while( swerve ) {
        if( crush.checkRadar() ) {
            crush.setSpin( 90 );
        } else {
            crush.setSpin( 0 );
        }
        pause( 0.1 ) ; }
    crush.setSpin( 0 ) ;
    crush.setRadarRange(0.0) ;
}
```

Drawing with the Turtle

- Some more turtle commands.
  - crush.penDown() ;
    - Causes the turtle to start drawing a line.
  - crush.penUp() ;
    - Causes the turtle to stop drawing a line.
  - crush.clearTrail() ;
    - Erases whatever the turtle has drawn (and stops drawing)

Drawing with the Turtle (cont)

- crush.setTrailLength() ;
  - Specifies a maximum length for the trail
- **boolean** b =

  crush.doesTrailIntersectTrail(t2);
  - t2 is another or the same turtle.
  - b will be true if the trails intersect.

Radar

- Each turtle has a very limited form of “radar”. The radar monitors a pie-slice shaped region in front of the turtle
- First you must set the range of the radar using

  crush.setRadarRange(r) ;

  where r is a radius measured in pixels. r should be at least 10.0 as that is the radius of the turtle.

Radar

- You may check whether any object is in the radar sector by calling

  crush.checkRadar()

  For example

  ```
  if( crush.checkRadar() ) {
      setSpeed(0.0) ; }
  ```

- You may check for a specific object

  crush.checkRadarFor(somethingElse)

  For example

  ```
  if( crush.checkRadarFor( squirt ) ) {
      increaseCrushsScore; }
  ```

- Note the radar only detects other objects, not the edges of the turtle arena

Radar (continued)

- You may change the direction that the radar points with

  ```
  crush.setRadarDirection( d ) ;
  ```

  where d is in degrees clockwise from the turtles current heading. The default is 0.0

- You may change the angle of the pie slice with

  ```
  crush.setRadarBeamAngle( a ) ;
  ```

  where a is in degrees. The default is 90.0.

Time

- pause(*t*)
- This command causes the TurtleController to pause for *t* seconds.
- Can you make a button to make the turtle draw a square by tracing out the shape of a square?

Start-up Tasks.

- You can give a list of TurtleController methods to be executed as soon as the program starts.
- These are listed in a list in the TurtleController called “startUp”
- For example the TurtleController’s “reset” method is executed at the start. See next slide.

An example start-up task.

```
public static String[] startUp = new String[] {
        "reset", …other start up task names go here… } ;

    public void reset() {
        crush.reset();
        crush.clearTrail() ;
        crush.setTrailLength( Double.POSITIVE_INFINITY ) ;
        double x = arena.getWidth() / 2.0 ;
        double y = arena.getHeight() / 2.0 ;
        crush.setPosition(x, y) ; }
```

On-going tasks

- A method that does not stop is “on-going”. For example:

```
public void bounce( ) {
    while( true ) {
        double rightEdge = arena.getWidth() ;
        double bottomEdge = arena.getHeight() ;
        double x = crush.getPositionX() ;
        double y = crush.getPositionY() ;
        …much omitted…
        pause(0.1) ;
    }
}
```

  - Any such on-going loop should include a short “pause”.

On-going start-up tasks.

- By adding an on-going task such as “bounce” to the startUp list, the method will be executing all the time.

Concurrency

- Multiple tasks may be active at a time. This is called **concurrency**.
- While the pause command is being executed, other tasks may change the state of the turtle world. For example

```
...
crush.setRateOfChangeOfSpeed( 0.0 ) ;
pause( 0.1 ) ;
double acceleration = crush.getRateOfChangeOfSpeed() ;
// acceleration may, or may not be 0.0.
...
```

Creating more turtles

- Initially the TurtleController creates one green turtle
- You can have it create more turtles
  - Add a new “private” field to the TurtleController
    - E.g. add a line

      private Turtle redTurtle = **new** Turtle(Color.red) ;;

      after the line that says “**private** Turtle turtle ;
  - In the “constructor” of the TurtleController add

    arena.add(redTurtle) ;

The Arena

- The Turtle Controller knows a GrobArena object by the name “arena” representing the area on which the turtles move.
- You may interrogate the arena with

```
double width = arena.getWidth() ;
double height = arena.getHeight() ;
```

Adding Turtles to the Arena

- You may add additional turtles
- To add another turtle.
  - In the TurtleController class add a variable declaration for the new turtle. E.g.:

```
private Turtle crush = new Turtle(Color.blue) ;
private Turtle squirt= new Turtle(Color.red) ;
private GrobArena arena ;
```

  - In the "TurtleController" method add lines as follows

```
public TurtleController( GrobArena arena, Log log ) {
   this.arena = arena ;
   this.log = log ;
   arena.add( crush ) ;
   squirt = new Turtle( Color.red ) ;
   arena.add( squirt ) ;.
}
```

New lines

Adding other objects to the Arena

- To add a rectangle to the arena.
- In the TurtleController method. Add these lines

```
// The constructor
    public TurtleController( GrobArena arena, Log log ) {
        this.arena = arena ;
        this.log = log ;
        arena.add( crush ) ;
        Rectangle rect = new Rectangle( 100, 100, 20, 30 ) ;
        ShapeGrob r = new ShapeGrob( rect ) ;
        arena.add( r ) ;
}
```

New Lines

The Log

- The TurtleController knows a “Log” object by the name of “log” representing an area on which you may output messages.
  - log.println(s)
    - s may be a string or a number
    - prints s and then starts a new line.
  - log.print (s)
    - s may be a string or a number
    - prints s