Greenfoot

Introducing Java With Games And Simulations

Michael Kölling
University of Kent

Birmingham, June 2009

University of Kent

Workshop material

- Download and install Greenfoot: www.greenfoot.org
- Download the "Little Crab" scenario (little-crab.zip) from www.greenfoot.org/workshop

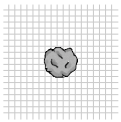
Wombats.

Object Orientation

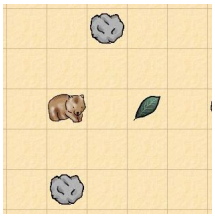
- Early understanding of key concepts is important
  - class
  - object
  - state
  - behaviour
- Not easy without tool support
- Most important: motivation

Asteroids, Ants and other creatures.

Variable resolution

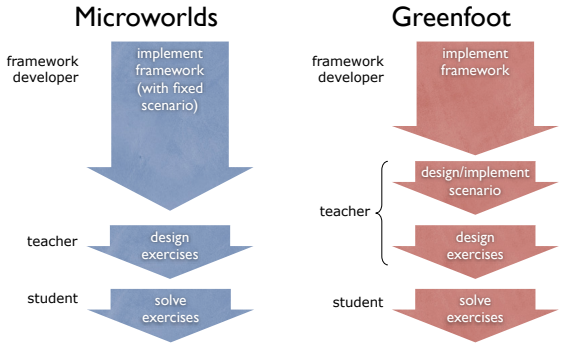


high resolution

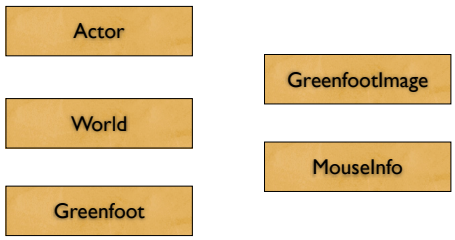


low resolution (tiled)

Division of work



Greenfoot classes



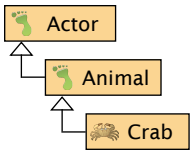
Actors

'Actors' have predefined state:

- image
- location (in the world)
- rotation

Actor methods

- act()



- getX(), getY()
- setLocation(int x, int y)
- ...

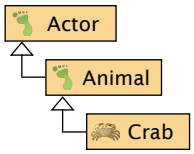
inherited from class 'Actor'

Animal

void **move()**

void **turn(int angle)**

boolean **atWorldEdge()**



Movement (1)

```
move();
```


Movement (2)

```
setLocation( getX()+1, getY() );
```

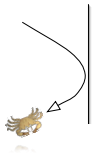

Key point

- You can choose the level of abstraction/ complexity to expose to your students by preparing the scenario.

"Little Crab"

Exercise:

- movement, edge detection, turning



Turn at edge

```
public void act()
{
    if(atWorldEdge()) {
        turn(17);
    }
    move();
}
```


Random behaviour

- Make the crab walk a little randomly (e.g. every N steps turn by M degrees, where N and M are somewhat random).

See:

```
Greenfoot.getRandomNumber()
```

Random walk

```
public void act()
{
    if(atWorldEdge()) {
        turn(17);
    }
    move();
    randomTurn();
}

public void randomTurn()
{
    if(Greenfoot.getRandomNumber(100) < 10) {
        turn(Greenfoot.getRandomNumber(90)-45);
    }
}
```


Worms...

- New subclass of *Animal:* Worm
- No behaviour needed
- Crabs eat worms... (*collision detection*)

See:

`canSee(Class clss)` in Animal
`eat(Class clss)` in Animal

Eating worms

```
if( canSee(Worm.class) ) {
   eat(Worm.class);
}
```


Key point (again)

- You can choose the level of abstraction/ complexity to expose to your students by preparing the scenario.
- *Animal* defines '*can*See' and '*eat*'...
- ...but it does not have to.

Crab

```
Actor worm = getOneObjectAtOffset(0, 0, Worm.class);
if (worm != null) {
    getWorld().removeObject(worm);
}
```


Crab

```
public void tryToEatWorm()
{
    if( canSee(Worm.class) ) {
        eat(Worm.class);
    }
}
```

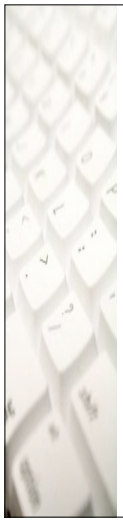
Crab

```
public void tryToEatWorm()
{
    if( canSeeWorm() ) {
        eatWorm();
    }
}
```


Simulation control

Exercise:

- Add a Lobster
  - Copy the functionality from crab
  - Lobsters hunt crabs, instead of worms
- The simulation should stop when the crab is caught.

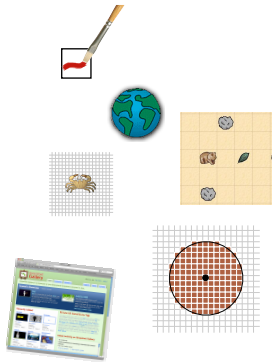


Keyboard input

```
if(Greenfoot.isKeyDown("left"))
    ...
```


Other topics

- working with images
- world methods
- world resolution
- collision detection
- export



World methods

- The world is an object, too.
- This object can also have methods.
- After every successful compilation, a world object is automatically created (using the default constructor).
- Useful for initialisation.

Initialisation

```
public CrabWorld()
{
    super(560, 560, 1);
    addObject ( new Crab(), 200, 300);
}
```

Greenfoot Gallery

- Share!

Collision detection

`List` **`getIntersectingObjects`**`(java.lang.Class cls)`
Return all the objects that intersect this object.

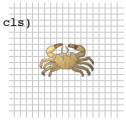
`Actor` **`getOneIntersectingObject`**`(java.lang.Class cls)`
Return an object that intersects this object.

`List` **`getObjectsAtOffset`**`(int dx, int dy, java.lang.Class cls)`
Return all objects that intersect the given location (relative to this object's location).

`Actor` **`getOneObjectAtOffset`**`(int dx, int dy, java.lang.Class cls)`
Return one object that is located at the specified cell (relative to this objects location).

`List` **`getNeighbours`**`(int distance, boolean diagonal, java.lang.Class cls)`
Return the neighbours to this object within a given distance.

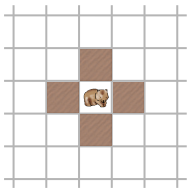
`List` **`getObjectsInRange`**`(int r, java.lang.Class cls)`
Return all objects within range 'r' around this object.



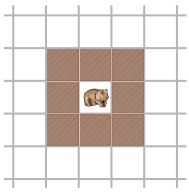
Neighbours

`List` **`getNeighbours`**`(int distance, boolean diagonal, java.lang.Class cls)`
Return the neighbours to this object within a given distance.

distance = 1



diagonal = false

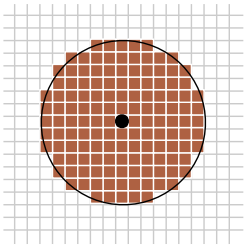


diagonal = true

Range

`List` **`getObjectsInRange`**`(int r, java.lang.Class cls)`
Return all objects within range 'r' around this object.

cell: 10 pixels



r = 6 (cells)

An object is "in range" if its centre point is inside the circle.

Looking forward...

- mobile phones
- Graphics program integration

More information

- www.greenfoot.org
- discussion group
- scenario repository
- tutorials (text and video)
- Greenfoot Gallery
- Michael: mik@kent.ac.uk