# 1 About the Limits of Turtle Animation

You can animate turtles using turtles created by the class `AniTurtle`. Our implementation has however a *big* limitation that we want to present you now. This limitation is especially important in the context of conditional loops and user interaction. We are really sorry to impose you this extra complexity but we could do it otherwise. We strongly recommend you to avoid using animated turtle in the context of conditional loops if you do not understand the following. We designed everything so that there is no influence. We thought even to remove the animation because of this limitation. We kept it because the animation is an important help for the beginning of the book and young novices. Therefore the animated turtle can be used freely in the previous chapters. However, when mixed with user interaction and conditional loops, it may block your system if you do not follow our warnings.

## 1 Do not Mix Traditional Conditional Loops And User Interaction

First you should not use directly reference to Sensor inside traditional conditional loop such as `whileFalse:`. Hence you have to transform for the method wandering shown below 1.1 and use `doUntilButtonPressed:` and `do:until:` as shown in the methods 1.2 or 1.3.

**Method 1.1**

```
Turtle>>wandering

   [Sensor anyButtonPressed] whileFalse:
      [self go: 30 atRandom.
      self turnLeft: 30 atRandom]
```

**Method 1.2**

```
AniTurtle>>wandering
   "AniTurtle new wandering"

   self doUntilButtonPressed:
      [self go: 30 atRandom.
      self turnLeft: 30 atRandom]
```

**Method 1.3**

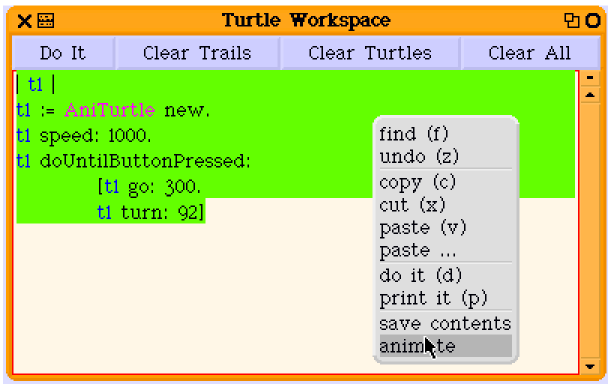
```
AniTurtle>>wanderingAni
   "Make the turtle walking by a random length and turn randomly its direction"

   self do: [self go: 30 atRandom.
      self turnLeft: 30 atRandom]
   until: [Sensor anyButtonPressed]
```

## 2 Always Execute Scripts in Forked Blocks

To execute the animation we have to surround the expression in a block and execute the block using the message `fork`. For example if we want to execute `AniTurtle new wandering` we write as `[AniTurtle new wandering] fork`.

Note that the environment, *i.e.*, the Turtle Workspace, provides a special menu item that give you the possibility to execute some code by wrapping it automatically in a block and sending the message fork to this block as shown by Figure 1.1. The script 1.1 shows an example of turtle animation with conditional.



```
| t1 |
t1 := AniTurtle new.
t1 speed: 1000.
t1 doUntilButtonPressed:
        [t1 go: 300.
        t1 turn: 92]
```

Figure 1.1: Executing an animation using the animate menu item.

WARNING. Not using `fork` or the animate menu item of the Turtle Workspace to execute expression containing conditional loops and user interaction conditions may block SQUEAK

**Script 1.1 (*Animating a Turtle)***

```
[| t2 |
t2 := AniTurtle new.
t2 speed: 10000.
t2 color: Color magenta.
t2 go: 100.
t2 turn: 90.
t2 direction = 90
   ifTrue: [t2 color: Color red].
t2 color = Color magenta
   ifTrue: [t2 color: Color green]] fork
```

## 3 Animating Multiple Turtles

Note that sometimes we would like to animate multiple turtle in parallel. This is possible but here you have to use the `fork` message explicitly as you have to decide what you want to animate. The script 1.2 shows how two turtles can be in different loops waiting to be stopped by a user interaction (See Figure **??**). Note that both loops are enclosed in forked blocks.

**Script 1.2 (*Animating two turtles)***

```
| t1 t2 |
World clearTurtleTrails.
Turtle deleteAllTurtles.
[ t1 := AniTurtle new.
t1
   doUntilButtonPressed: [t1 go: 300.
      t1 turn: 92]] fork.
[ t2 := AniTurtle new.
t2
   doUntilButtonPressed: [t2 color: Color magenta.
      t2 extent: 200 @ 200.
      t2 extent: 100 @ 100.
      t2 penSize: 1.
      t2 penColor: Color brown.
      t2 go: 300.
      t2 turn: -92.
      t2 lookLikeTriangle.
      t2 go: 300.
      t2 extent: 50 @ 50.
      t2 turn: -92.
      t2 color: Color red.
      t2 go: 300.
      t2 turn: -92.
      t2 penSize: 3.
      t2 penColor: Color green.
      t2 lookLikeCircle.
      t2 go: 300.
       t2 turn: -92]] fork
```

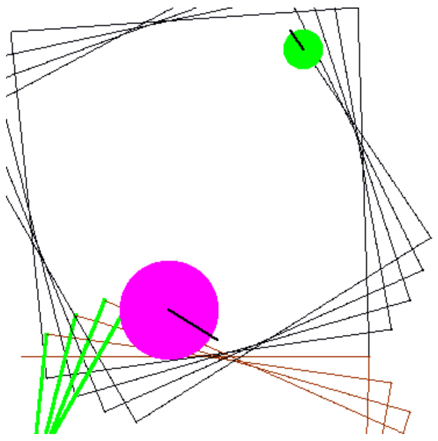


Figure 1.2: Two turtles animated.

## Summary

If you did not understand what we just presented, please do not use the animated turtle in the context of conditional loops.

Not using `fork` or the animate menu item of the Turtle Workspace to execute expression containing conditional loops and user interaction conditions may block SQUEAK