

CompSci 94

Classwork: Writing Functions

October 8, 2020



Prof. Susan Rodger

CompSci 94 Fall 2020

1) Overview of story

- We have a cat who loves to jump, and wants to jump over three creatures at a time. The cat will jump over them over and over again, even though the creatures can change their size and get taller! We also have two characters a tortoise and a penguin who like to argue about lots of things, like who is closer to the table!
- **Follow the steps** to write functions to help create this program

2) Setting up the scene

- **Use the starter world** on the calendar page called `catJumpingOverThreeStarter.a3p`.
- Or Follow these instructions to build it.
 - Add in any ground cover, I picked desert
 - Drag in these five objects placed as they look:
 - Quadruped: `AbssyinianCat` (rename `Cat`), coyote
 - Biped: tortoise
 - Flyer: penguin
 - Prop: `coffeeTable`
 - See pictures on next few slides

Starting setup

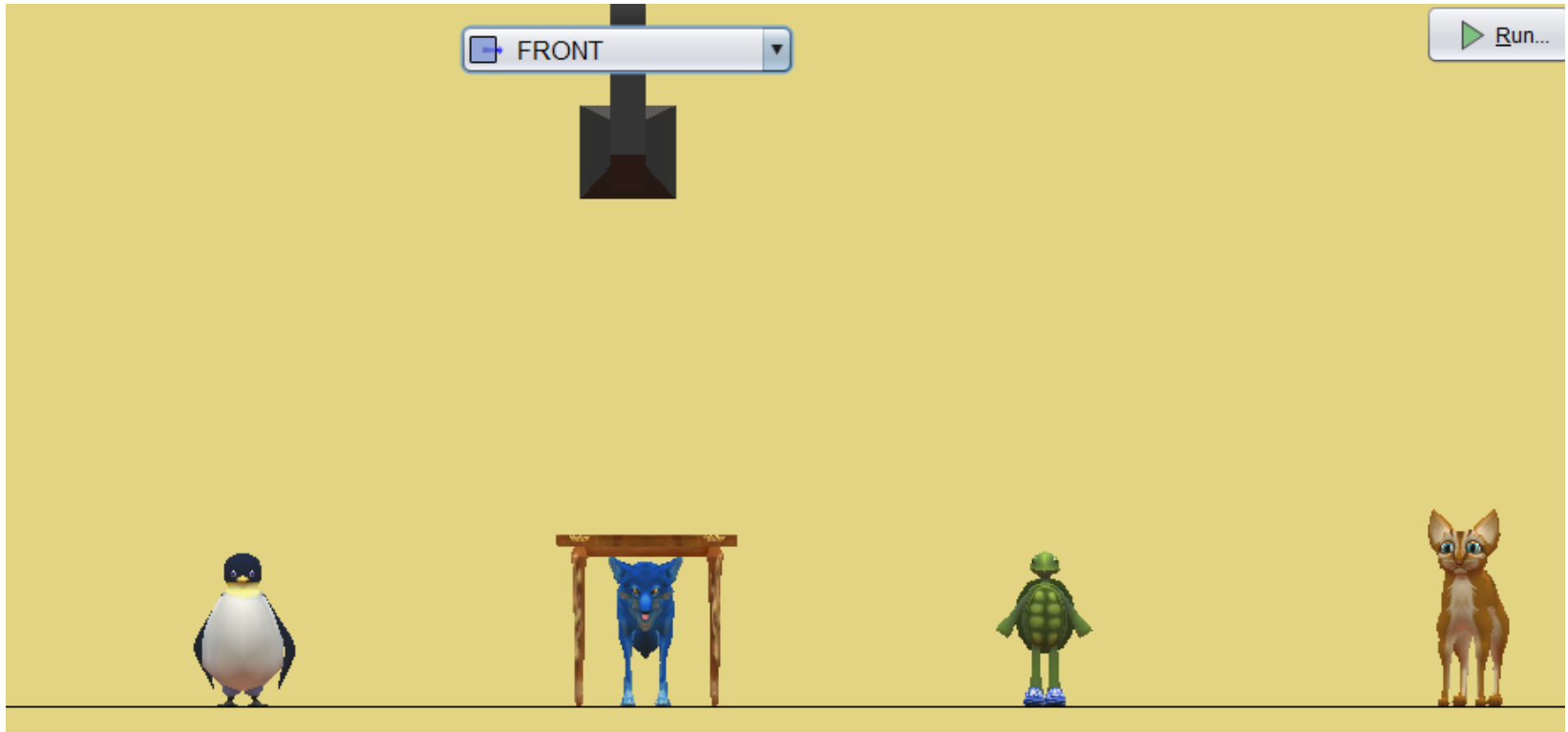
- Spread animals out, but in a line.



Sideview – all lined up



Frontview



3) Teach the cat a fancy jump

- Note: The starter program already has the cat jump procedure! **(IGNORE this step and go to step 4) if you use the starter program)**
- Write a **cat jump procedure** (not a function) (see picture next slide)
- The cat should move up
- As the cat moves forward its front legs, tail and back legs should turn like it is stretching and then back to their normal position.
- The cat should move down the same amount
- Be sure to have **parameters** for how high and how far it goes
- Note the parameters are different than what we had with the dalmatian jump

Jumping over

- Note legs stretched out and tail too!



4) Test the Jump procedure

- In myFirstMethod, first put in a do in order.
- Have the cat turn to face the tortoise, and then jump over all three of the other creatures, using the jump procedure
- For now, just guess how high and how far it has to jump.

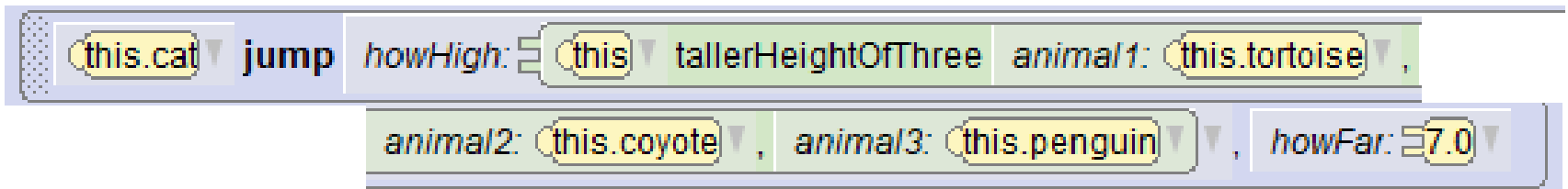
4) Write the **Scene** function *tallerHeightOfThree*

- This **function** has **three** parameters of type **SJointedModel** and returns the **height** (a decimal number) of the tallest of the three objects.

```
declare DecimalNumber function tallerHeightOfThree with parameters: SJointedModel animal1 ,  
                                                                    SJointedModel animal2 , SJointedModel animal3
```

5) Test your tallerHeightOfThree in myFirstMethod

- In myFirstMethod where the cat jumps over the three, replace the “how high” to jump with a call to tallerHeightOfThree method.



```
this.cat.jump(howHigh: this.tallerHeightOfThree(
    animal1: this.tortoise,
    animal2: this.coyote, animal3: this.penguin),
    howFar: 7.0)
```

The screenshot shows a code editor with a snippet of Java code. The code is for a method call on `this.cat` with the method `jump`. The first argument is `howHigh`, which is assigned the value of `this.tallerHeightOfThree` followed by three arguments in parentheses: `animal1: this.tortoise`, `animal2: this.coyote`, and `animal3: this.penguin`. The second argument is `howFar`, which is assigned the value `7.0`. The code is highlighted in a light blue box.

- Make sure your program still runs and the cat jumps exactly the height of the tallest object.

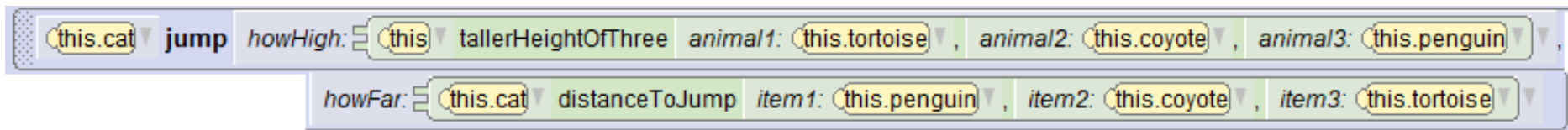
6) Write the **cat** function **distanceToJump**

- Make sure you are writing a **function**, a **cat function**
- Create three **parameters**, all of type **SJointedModel** and return the **distance needed** (a decimal number) to jump over all three.
- This function first calculates which animal is the furthest away and then calculates and returns the distance needed to jump over that animal. (hint: besides distance to that animal, consider some dimension of that animal and a dimension of the cat)

```
declare DecimalNumber function distanceToJump with parameters: SJointedModel item1 ,  
                                                                SJointedModel item2 , SJointedModel item3
```

7) Now fix the jump call in myFirstMethod to use the DistanceToJump function

- Now you can calculate exactly how far to jump. Fix the cat jump procedure to call distanceToJump for howFar to jump.



- Make sure your cat jumps far enough

8) Add more code to myFirstMethod

- After jumping, have the cat turn to face the tortoise.
- Have the tortoise resize by 2.
- Then have the cat jump over all three animals again, calling **tallerHeightOfThree** to figure out how high to jump, and distanceToJump for how far.
- Now have the cat turn again to face the tortoise.

8) Add more code (cont)

- The penguin resize by 3.
- Have the penguin **move right** 1.
- Have the cat jump over them again
- Then cat turn to face the tortoise
- coyote resizes by 4.
- Have the tortoise **move left** by 2.
- The cat jumps over them again.
- Run your program to see the cat jumping over animals increasing in their size.

9) Write the **Scene** function *furthestFromTable*

- This function has two parameter objects of type **SJointedModel** and returns the object (also of type **SJointedModel**) that is furthest from the coffeeTable

```
declare SJointedModel function furthestFromTable with parameters: SJointedModel some1 ,  
                                                                    SJointedModel some2
```

10) Add more code in myFirstMethod

- Add this code at the end of myFirstMethod.
- Test the **furthestFromTable** function with the **tortoise and penguin** who like to argue about everything. Have the object that is furthest from the table (tortoise or penguin) say “I’m furthest from the table” (you do this with just one statement that uses your furthestFromTable function to calculate which one is the furthest, you do not need an if statement here!)

10) Add more code (cont)

- Have the object that is furthest from the table turn to face the table (this is also one statement only and uses your furthestFromTable function)
- Then have the object that is furthest from the table move towards the table 2 units. (this is one move statement)
- Now Put these three statements in a Count 6 loop. What happens?

11) Write a **Scene** function named **randomEndPhrase**

declare TextString *function* **randomEndPhrase** Add Parameter...

- This function returns **one of four** random phrases for the ending of this project. The return value should be a TextString. For example, one of them might be “That’s all folks!”
- Note that there are **NO** parameters. You should randomly select a phrase.

Test out your new function.

- At the end of myFirstMethod, have the penguin say a random ending. And then say a second random ending (may or may not be the same one.)
 - The phrase should be selected randomly.
 - Hint: Generate a random number and store it in a variable, then compare the number to decide which phrase to return.

12) Save the cat as a class

- Save the cat as a class (.a3c file) so you will have it for other alice programs.