CSCI 1106
Lecture 7

Variables and Threads

Announcements

- Today's Topics
  - The need to remember
  - Variables
  - Types of Variables
  - An example of using variables
  - Multitasking
  - Multithreading
  - Interference

A Light Problem

- Task: Write a program that moves the Tribot towards a light source
- Approach:
  - Loop
    - Turn tribot in a circle until it faces the brightest point in the room
    - Drive forward a short distance
- Q: How do we determine the brightest point?

The Need to Remember

- Observations:
  - To determine the brightest point, need to compare it to other points
  - To compare to other points, need to remember the brightness of those points
  - Need to remember (store) previously measured brightness
- Most programs need to store various values
- Programs use a *variable* to store a value

Variables

- A *variable* is a ``container'' to store a value
- Supports two operations:
  - *Write*: store a value in the variable
  - *Read*: retrieve a value from the variable

- A variable is identified by its name
  - The name should reflect what the variable is used for
  - E.g., brightness
- A variable has a *type* restricting the kind of values that it can store

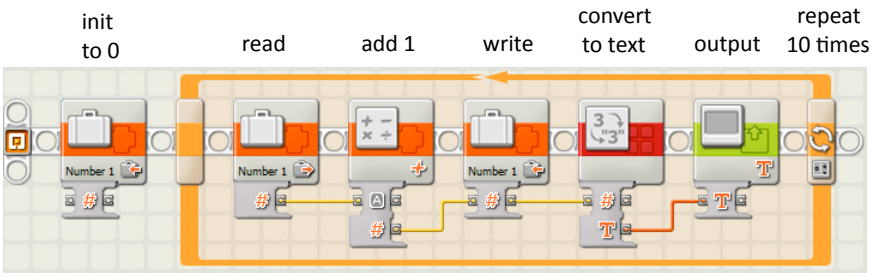
Variable Types

- Observation: Some containers can store only specific items
- Key Ideas:
  - The *variable type* determines the kinds of values that a variable can store
  - The type of value must match the type of variable
- Variable Types
  - Numbers: Decimal values, e.g., 1.25, 3.14, 0, 42
  - Text: e.g., "Hello, World!", "Goodbye", "woot"
  - Logical: Boolean values, e.g., *true*, *false*
- Remember: You cannot store text in a number variable!

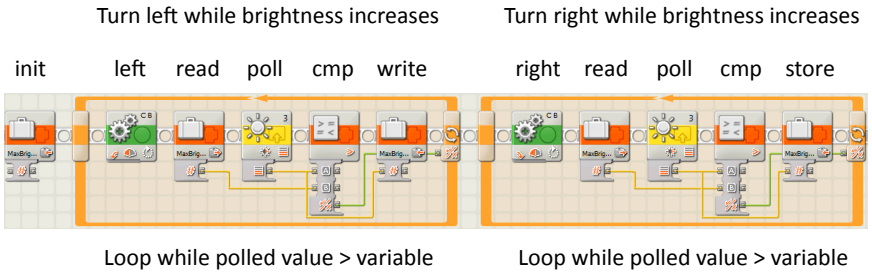
Using Variables

- Create the variable:
  - Edit ➔ Define Variable ➔ Create
  - Enter name of variable
  - Specify type (Number, Text, or Logical
- Use the variable block to read or write the variable
  - Place block
  - Specify variable to use
  - Specify operation
  - Connect wires to read/write value
- Note: wires must have the same type as the variable

Count to 10 and Display



Example: Seek the light!

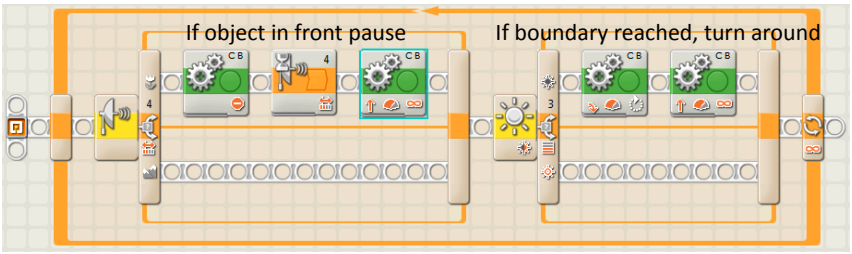


Multitasking

- *Multitasking* is the act of doing multiple tasks concurrently
- Examples of multitasking
  - Walking down the street and texting
  - Driving and talking
- Observation: Each task consists of its own sequence of operations

A Multitasking Task

- Task: Write a program so that the tribot
  - Avoids the boundary (black line)
  - Pauses if there is an object in front of it
- One Approach: One big loop



Problems with the Approach

- Tribot can only do one thing at a time
- Increased response time
- Unnatural code structure
- Increased code size and complexity

- Question:
  - Can we do better?

- Idea: Implement multitasking with *multithreading*

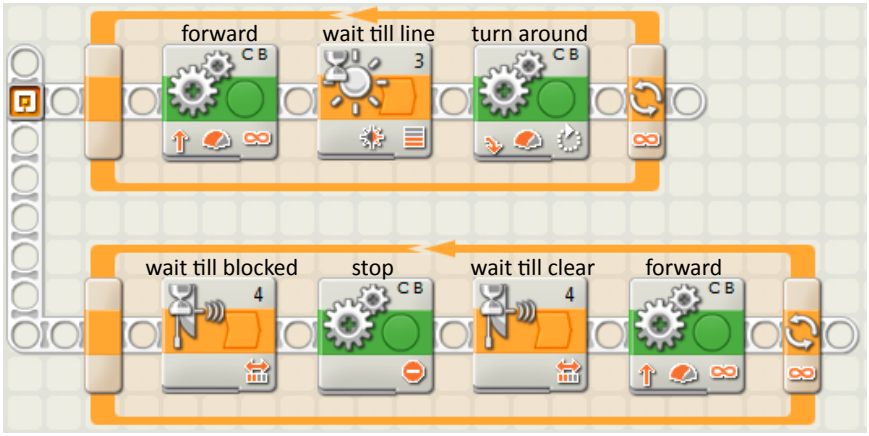
Threads

- A *thread* is the execution of a sequence of instructions
- All our programs so far have consisted of one thread
- All our programs had one girder

Multithreading

- *Multithreading* is the concurrent execution multiple sequences of instructions
  - Each thread is represented by a girder
  - Multithreaded programs have multiple girders
- Idea: Use multithreading to implement two tasks:
  - Avoid boundary
  - Pause if object is ahead

The Multithreaded Approach



Interfering Threads

- Question: What happens if multiple threads try to control the motors at the same time?
- *Interference* occurs when a thread violates another thread's assumptions
  - E.g., Only it has control of the motors
- Question: How do we prevent this?

Preventing Interference

- Only one thread controls a given actuator
  - I.e., only one thread may control the motors
- Threads must coordinate their behaviours
  - E.g., both threads may control the motors but not at the same time
- How do threads coordinate? Via variables!

A Solution to Interference