preopCreate Python Module

March 31, 2011

1 Introduction

preopCreate is a wrapper for the pycreate library to make the function calls more similar to those in PREOP. The pycreate library was modified to utilize the pybluez library, thus allowing a bluetooth connection to the robots.

For more information on PREOP, please visit **the PREOP website**.

For more information on pycreate and pybluez, please contact the respective institutions that developed those libraries.

1.1 Contact Information

If you're having problems using preopCreate, or need to obtain another copy of the files, please contact Agata Kargol at **aukargol@crimson.ua.edu**.

2 Installation Instructions

2.1 System Requirements

This module is compatible with Python 2.6.

The module depends on the pycreate and pybluez modules. You must have both installed for preopCreate to function.

2.2 Installation Procedures

Installing under Windows is very simple and only involves some copying an pasting. However, first you will need to find your Python26 folder. It is most likely located under **C:\\Python26**.

Copy and paste everything in the **createMods** folder on your flash drive to **C:\\Python26**.

Copy and paste everything in the **bluetoothMod** folder on your flash drive to **C:\\Python26\Lib\site-packages**.

Congratulations! You have now installed preopCreate, pycreate, and pybluez.

3 Using preopCreate

preopCreate defines a robot class that you may utilize to connect to your robot. You just need to call it. The available function calls and examples of code are documented below.

Several constants have already been defined for you. These global constants are defined in all caps, so please follow capitalization rules.

3.1 Importing the Module

Before you can use the module, you will need to include the following line of code:

```
from preopCreate import *
```

If you run your script at this point, you should get no errors. If you get an error, then the module was not correctly installed.

3.2 Creating a Robot

To create a robot, you will need the 4 digits off the BAM module connected to your robot. For this example, assume those digits are **4AB3**. To create a robot, you would make the following call:

```
marvin = robot_create("4A:B3")
```

Please note that you HAVE to add the colon in between the 2nd and 3rd digits as shown above. This is probably the only difference between this module and PREOP.

You may now use the variable name **marvin** to refer to your robot. If you name your robot something else, please remember to call that name and not **marvin**.

3.3 Robot Function Calls

3.3.1 Moving the Robot

MOVE To move a robot forward, you will need to call the **move** function. **move** takes 2 or 3 parameters: direction of travel, amount (in meters), and duration (in seconds). Please note that duration is optional; as with PREOP, the function will assume you mean to use 1 second as your duration if you do not enter anything.

```
marvin.move{FORWARD, 1)
marvin.move{BACKWARD, 2, 2)
```

MOVE AT SPEED PREOP also allowed you to tell the robot to move at a certain speed rather than a certain amount. To do this, you will need to call the **moveAtSpeed** function. **moveAtSpeed** takes 2 or 3 parameters: direction of travel, speed (in meters per second), and duration (in seconds). Please note that duration is optional; as with PREOP, the function will assume you mean to use 1 second as your duration if you do not enter anything.

```
marvin.moveAtSpeed{FORWARD, 0.5)
marvin.moveAtSpeed{BACKWARD, 0.25, 0.5)
```

3.3.2 Turning the Robot

TURN To turn a robot, you will need to call the **turn** function. **turn** takes 2 or 3 parameters: direction of travel, amount (in revolutions), and duration (in seconds). Please note that duration is optional; as with PREOP, the function will assume you mean to use 1 second as your duration if you do not enter anything. Also note the direction is relative to the robot, so **RIGHT** would be equivalent to clockwise, and **LEFT** equivalent of counter-clockwise.

```
marvin.turn{LEFT, 0.25)
marvin.turn{RIGHT, 0.5, 2)
```

TURN AT SPEED PREOP also allowed you to tell the robot to turn at a certain speed rather than a certain amount. To do this, you will need to call the **turnAtSpeed** function. **turnAtSpeed** takes 2 or 3 parameters: direction of travel, speed (in revolutions per second), and duration (in seconds). Please note that duration is optional; as with PREOP, the function will assume you mean to use 1 second as your duration if you do not enter anything. Also note the direction is relative to the robot, so **RIGHT** would be equivalent to clockwise, and **LEFT** equivalent of counter-clockwise.

```
marvin.turnAtSpeed{LEFT, 0.5)
marvin.turnAtSpeed{RIGHT, 0.25, 0.5)
```

3.3.3 Lights

You have access to two LEDs on the top of your robot: the Advance LED and the Play/Power LED.

POWER LED COLOR To set the Play/Power LED color, you will need to call the **powerLEDColor** function. **powerLEDColor** takes only one parameter: an integer from 0-255. 0 will make the LED turn green, 255 will make it turn red. Note that you will need to turn the lights on to see the color.

```
marvin.powerLEDColor(0)
marvin.powerLEDColor(255)
```

PLAY LED To turn the Play/Power LED on, you will need to call the **isPlayLEDOn** function. **isPlayLEDOn** takes one parameter: **True** or **False**. **True** will turn the LED on, **False** will turn it off.

```
marvin.isPlayLEDOn(True)
```

ADVANCE LED To turn the Advance LED on, you will need to call the **isAdvanceLEDOn** function. **isAdvanceLEDOn** takes one parameter: **True** or **False**. **True** will turn the LED on, **False** will turn it off.

```
marvin.isAdvanceLEDOn(True)
```

3.3.4 Sensors

PREOP also gives you access to several of the robot's sensors. Each of the sensors is accessible through functions which are detailed below. All of the sensor function return **True** or **False**. This allows you to have the following code:

```
if ( marvin.rightBumperIsDepressed ) :
marvin.move(BACKWARD, 0.1)
marvin.turn(LEFT, 0.5)
if ( marvin.leftBumperIsDepressed ) :
marvin.move(BACKWARD, 0.1)
marvin.turn(RIGHT, 0.5)
marvin.move(FORWARD, 0.2)
```

WHEEL DROP SENSORS To determine whether the robot's left wheel, right wheel, or caster wheel is dropped, you can call the following functions:

```
marvin.leftWheelIsDropped()
marvin.rightWheelIsDropped()
marvin.casterWheelIsDropped()
```

BUMPER SENSORS To determine whether the robot's right bumper or left bumper is depressed, you can call the following functions:

```
marvin.rightBumperIsDepressed()
marvin.leftBumperIsDepressed()
```

BUTTON SENSORS To determine whether the robot's play/power or advance buttons are pressed, you can call the following functions:

```
marvin.playButtonIsPressed()
marvin.advanceButtonIsPressed()
```

WALL DETECTION To determine whether the robot detects a wall, you can call the following function:

```
marvin.detectsWall()
```

3.3.5 Playing Notes and Songs

Playing notes and songs requires that you create notes and songs using the **note** and **song** classes. Both are described in the next section. For the example code below, assume you have a note named **note1** and a song named **song1**.

```
marvin.playNote(note1)
marvin.playSong(song1)
```

3.4 Creating Notes and Songs

NOTES PREOP required you to create a note with the big menu where you specified the note, the octave, the tempo, and the length of the note. This module does very much the same thing. You have to specify the note, (for which you can use the values specified in the next paragraph) the octave, (has to be between 2-8) the tempo (in beats per minute), the length of the note (in beats), and the plus (in beats). If you do not specify the tempo, a tempo of 120 is assumed. If you do not specify the length of the note, a length of 1 is assumed. If you do not specify the plus duration, a plus duration of 0 is assumed. Note that you must specify the length if you specify a plus duration, and you must specify a tempo to specify a note length.

The values that you can use to specify the note are **C**, **C_SHARP**, **D**, **D_SHARP**, **E**, **F**, **F_SHARP**, **G**, **G_SHARP**, **A**, **A_SHARP**, and **B**. Thus, all of the following examples are valid ways of creating a note:

```
note1 = note(C, 2)
note2 = note(D_SHARP, 4, 125)
note3 = note(F_SHARP, 0.5, 62, 4)
note4 = note(G, 1, 120, 1, 0.5)
```

SONGS Songs are defined as a list of notes. To create a song, you must first create it, as shown below.

```
song1 = song()
```

You must then add notes to this song. If you have already defined your notes as shown below, you may add notes to your song by calling the **addNote** function, which takes a note as a parameter: (Using the notes defined above)

```
song1.addNote(note1)
song1.addNote(note2)
song1.addNote(note3)
song1.addNote(note4)
```