CS4303 Video Games:
`Space.Trash` Players Guide

December 2019

Introduction

Welcome to `Space.Trash`! You are a factory worker in a distant future Earth where most programming languages have been lost to time as systems continued to evolve. Long ago, on New Years of the year 2020, Python 2.7 was officially retired! However, many projects refused to update to the newer Python 3. Years later, Python 2.7 was used to develop the first general robotic programming frameworks - `PRPF`. `PRPF` became popular and quickly became the *only* robotic programming framework to exist; crowning Python 2.7 the most used programming language ever.

Python Robotic Programming Framework

`PRPF` is the industry standard for robotic control. `PRPF` makes programming any robot simple with plug & play bindings to hardware and full[1] support for Python 2.7. `PRPF` does not include any text editors, we recommend enterprises use `G4PRPF` for easy integration.

Functions

Upon installation of `PRPF`, the programmer instantly gets access to the following functions:

- `move(x)` - Moves a robot `x` meters ahead in the direction it is facing
- `left(degrees)` - Turns the robot left by `degrees` degrees
- `right(degrees)` - Turns the robot right by `degrees` degrees
- `align()` - Aligns the robot with the closet cardinal direction
- `attack()` - The robot spins to attack with it's inbuilt weapon
- `hold()` - The robot attempts to pick up an item it is on top of
- `interact()` - The robot attempts to interact with a container it is on top of
- `isNear()` - Returns a boolean specifying if the robot is near to any lifeforms
- `canMove(x)` - Returns a boolean specifying if the robot can be in the position `x` meters away
- `getX()` - Returns the robots X position to the nearest meter
- `getY()` - Returns the robots Y position to the nearest meter
- `getDirection()` - Returns the robots direction in degrees

A `PRPF` subscription can be purchased for only $4000^{\diamond}$ monthly for access to more functions.

[1] Support can be limited within the `PRPF` installer

G4PRPF

G4PRPF is a free editor for enterprise installations of PRPF. G4PRPF offers many solutions for your business - an inbuilt console, default code for important sections, and with the purchase of G4PRPF^{PRO}, integration with code-sense tools.

Layout

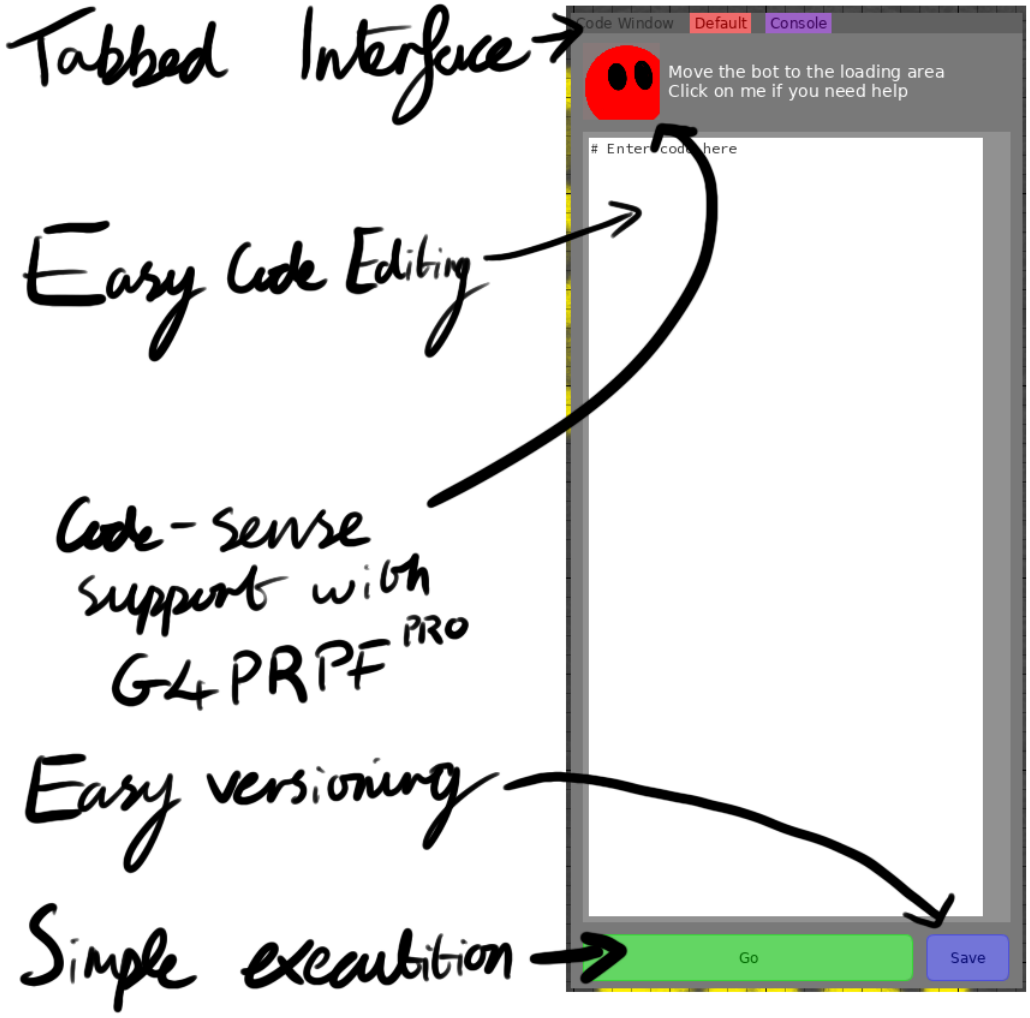

Figure 1: G4PRPF for Embedded Devices

Tabs within G4PRPF for Embedded devices will fill their parent panel when opened by clicking on the tabs name, simply click on the tabs name again to minimise the tab. The console within G4PRPF shows errors and prints out any `print()` statements. The console will blink green whenever text is printed, and red whenever an error is encountered.

Security

PRPF is a very open framework, allowing functionality with all of Python 2.7. Your organisation might want to disallow some of this functionality to prevent employees writing malicious code, System admins can change what is disallowed in the G4PRPF installer. G4PRPF disallows the following by default:

- `import`
- `open`
- `breakpoint`
- `exec`
- `input`
- `file`

G4PRPF also disallows statements that would override internal variables, however these are not listed.

Space.Trash

Space.Trash consists of many levels where you need to program bots within G4PRPF using PRPF. Solutions to these levels are not given within this guide, one can however refer to **Python Programming** to brush up on programming with Python.

Elements

The elements of a normal level are shown in Figure 2: Other level elements will appear and their function be

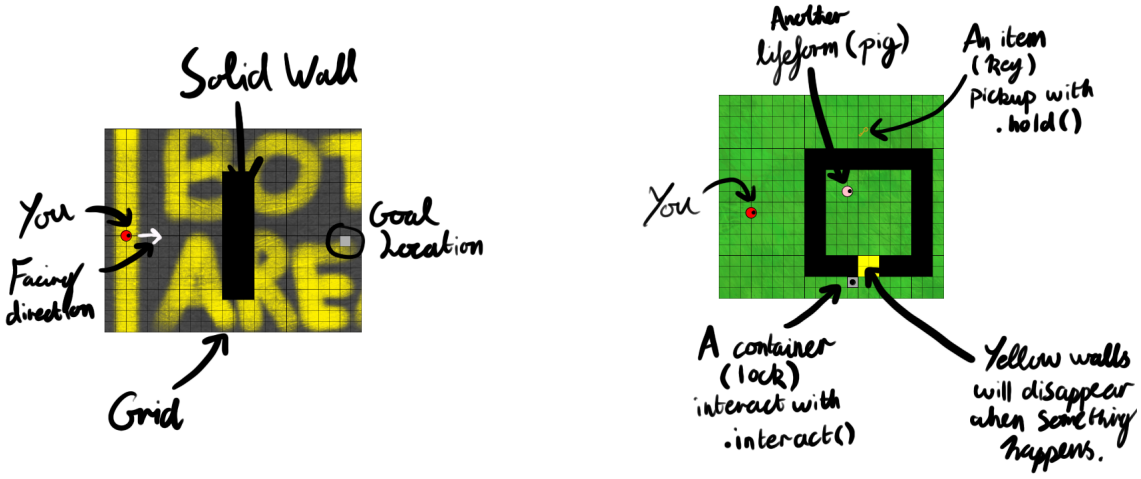

Figure 2: Showing off elements of a level

be referenced within the game.

Python Programming

Variables

In Python, set a variable with the equality operator `var = 3`, and compare variables with comparison operators `myvar < 3`.

If statements

In Python, one can define an `if/elif/else` statement that will execute the body of the `if` statement if the condition is true. Otherwise, it will execute the body of the `elif` statement if it's condition is true. Otherwise, it will execute the body of the `else` statement:

```
if (1 < 2):
  # do something
elif (3 > 4):
  # do something else
else:
  # do something else
```

While loops

In Python, one can define an `while` loop that will continuously execute the body of the `while` statement if the condition is true.

```
while (myvar):
  # do something
```

Functions

A function can be defined like the one shown below. Within a function, one can refer to params as a variable within the function body

```
def function(params):
  # function body
```

Classes

Making a class is easy. Simply define the class:

```
class robot():
  def __init__(self, name):
    self.name = name
  def getName(self):
    return self.name
```

We can see the constructor method `__init__(self)` which is called whenever an instance of the class is made (`myrobot = robot('Bob')`. Here, one argument is taken and that is used to build an attribute of the class `name`, it can be referred to with `self.name` within the class. A method is also defined, which can be called on an instance of the class `myrobot.getName()`.

Compiling & Running

Compiling and running the game vary across Operating Systems, the instructions given below are for Linux. Please see `README.md` to refer to commands for Windows. To compile the game & run the game, execute the following from the `src` directory:

```
find -name "*.java" > sources.txt
javac -cp ../libs/core.jar:../libs/G4P.jar:../libs/sound.jar:../libs/javamp3.jar
:../libs/json.jar:../libs/jsyn.jar:../libs/jython.jar @sources.txt
cp -a ../data data
java -cp .:../libs/core.jar:../libs/G4P.jar:../libs/sound.jar:../libs/javamp3.jar
:../libs/json.jar:../libs/jsyn.jar:../libs/jython.jar Start
```

To later remove the compiled `.class` files, execute the following from the `src` directory:

```
rm -rf data
rm -rf *.class
rm sources.txt
```