# Student Robotics 2020 Microgames Solutions

## Intro.1. Battery Charging 🔋

In this task, students are asked to start charging their batteries with the chargers which come in their robot kits.

Relevant docs section: (studentrobotics.org/docs)
**Kits/Batteries/HKE4 Charger**
Or
**Kits/Batteries/IMAX B6 Charger**
(Depending on which charger they have)

Ensure the students have followed the instructions for charging a battery using their model of charger. **Make sure they're using the fire-safe bags.**

## Intro.2. Assemble your kit 🏗

In this task, the students need to assemble their robotics kits. There's a video on the docs which has subtitles.

Relevant docs section: (studentrobotics.org/docs)
**Kits/Assembly**

This involves plugging all the boards to the brain board (white case) using USB, and most boards need power cables made up, and plugged into the power board (the one with a fan) using the green 'camcon' connectors. Make sure they have the polarity correct.

The one thing teams tend to forget is a single green connector next to the power button, which has a loop of wire from one socket to the other.

**If the students ask you anything you don't know, go through the docs with them. (This advice applies to all other tasks)**

# Intro.3. Find the Password 🔒

In this task, the students need to run the code on their robot.

They need to sign in to our code editor with their accounts. https://studentrobotics.org/ide/ the teacher should have received an email with the login details, and the ability to make an account for the student.

If they can't edit code, it's because they logged in with the teacher's account

Once they've edited the code, they click 'Export' then put the resultant robot.zip in the USB stick, plug it in, and turn on the robot.

They then press 'start' when its flashing.

If the code ran, there should now be a log.txt file in the USB with the secret code.

## *Solution:*

"`G0LD+51LV3R`"
Should be printed to the log file on their USB. Bonus points if they edited the code to print it 10 times.

# Silver.1. Wave the robot arm 👋

In this task they need to program their robot to wave a servo.

Relevant docs section: (studentrobotics.org/docs)
**Programming/sr/Servos**
**(and maybe Kits/Servo Board)**

Check their code, check they're setting the servos something between -100 and 100, and that they actually use time.sleep to give the servo time to move.

Also check that the servo is plugged in the right way around, the darker wire should be facing the bottom of the servo board (board with a grid of pins on the side). Check lights are on on the servo board too.

Solution:

The arm on the provided servo should move from side to side once the start button has been pressed.

# Silver.2. Post on the Forum 💌

The forum is at studentrobotics.org/forum, they log in with the same details as the IDE.

Solution:

There should be a post on the forum from a member of the team (not a team leader). If the post is inappropriate, please inform their team leader.

# Silver.3. Use the robot simulator 🤖

In this task the students are told to install the robot simulator and get it running. It's a bit of an old piece of software, and for reliability it comes with an entire copy of Python in a directory.

Relevant docs section: (studentrobotics.org/docs)
**Programming/Simulator**

The students can write their code in the PyScripter editor we give them, save it, and run it using run.py. The instructions on the docs are more detailed.

Solution:

They should write some code to make their robot drive in a circle. The circle can be of any size, as long as it's obvious the robot isn't simply spinning around on the spot. (they've set the power for more than 1 motor).

This boils down to:

```
From sr.robot import *

R = Robot()
R.motors[0].m0.power = 100 # or some number.
R.motors[0].m1.power = 20 # or some different number.
```

It's easy for students to get sidetracked on this task, so make sure they move on once they've got the robot to turn in a circle. There's an advanced simulator task in red 'Platform' task sheet.

# Silver.4. Bug Hunting 🐜

In this task the students are given some code, they need to find the bugs!

This is a big step up in complexity from the previous task, and our students come from very varied backgrounds, so you're expected to help them out a lot here.

First thing to do is tell them to copy/paste it into the simulator, then explain the errors as they come.

There are 10 errors in total, some are pretty tricky!

**Solution:**

```
from sr.robot import *
Import time # Should be lowercase I: 'import'

R = Robot # Should be a function: 'Robot()'

def drive(speed, time): # Second parameter can't be named 'time', it
means time.sleep doesn't work.
    R.motors[0].m0.power = speed
    R.motors[0].m1.power = speed
    time.sleep(time)
    R.motors[0].m0.power = 0
    R.motors[0].m0.power = 0 # m0 should be m1

def look_for_golden_token():
   markers = R.see() # Indentation error.
    arena_markers = []
    for m in markers:
        if m.info.marker_types = MARKER_TOKEN_GOLD: # Should be
`marker_type` ALSO `=` should be `==`.
            arena_markers.append(m)
    return arena_markers

while True:
    markers = look_for_golden_token(R) # This shouldn't have R as a
parameter.
    # Drive forwards if a token is seen
    if len(markers) != 0:
        if markers[0].distance > 0.5: # This should be `dist`, not
`distance`, check the docs!
            drive(20, 0.2) // Go forwards # Comment should start with
`#`, not `//`
```

***Fixed code:***

```
from sr.robot import *
import time

R = Robot()

def drive(speed, secs):
    R.motors[0].m0.power = speed
    R.motors[0].m1.power = speed
    time.sleep(secs)
    R.motors[0].m0.power = 0
    R.motors[0].m1.power = 0

def look_for_golden_token():
    markers = R.see()
    arena_markers = []
    for m in markers:
        if m.info.marker_type == MARKER_TOKEN_GOLD:
            arena_markers.append(m)
    return arena_markers

while True:
    markers = look_for_golden_token()
    # Drive forwards if a token is seen
    if len(markers) != 0:
        if markers[0].dist > 0.5:
            drive(20, 0.2) # Go forwards
```

# Gold.1. Lift Off 🚀

This task gets the students acquainted with the 'beep' function in their code, along with a quick intro to `time.sleep()` in case they're new to Python.

Relevant docs section: (studentrobotics.org/docs)
**Programming/sr/Power (bottom of page)**

Ideal code (using loops). If they don't use a loop, show them:

```
import time
from sr import *

R = Robot()

for i in range(5):
    # Beep for 0.5s
    R.power.beep(500, note='d')
    time.sleep(1)

# Beep for 2s
R.power.beep(2000, note='d')
```

Once the start button is pressed, the buzzer should beep 5 times, with a 1 second gap between, and a longer buzz at the end. Have a look at their code if there's any problem.

# Gold.2. Vision 👀

Teams are tasked with figuring out the numbers of the markers they were given. You should encourage them to write some code to automatically decode each letter, instead of working it out on paper.

Relevant docs section: (studentrobotics.org/docs)
**Programming/sr/Vision (entire page is relevant)**

The token numbers on the flags from left to right are:

`11, 8, 5, 19,`

which spells '**L' 'I' 'F' 'T', ('LIFT')**

If the students have 'KHES', that means they're off by 1. (They assumed 1 = 'a' , not 0).

# Gold.3. Wifi Zone Control 📶

This task has the students connect their phones to the wifi network hosted by the robot, then use the web interface to change a debug setting which tells the robot which corner of the arena it starts in.

**Not relevant to this task, but useful for when they actually build the robot:** Make sure the student's know that the robot zone does NOT change the numbering of the markers, this is logic they need to write themselves.

Relevant docs section:

**Programming/sr (bottom of page)**

The docs are quite hidden away and don't have an example in this case.

The usage is very simple:

```
from sr.robot import *

r = Robot()
zone = r.zone() # either 0, 1, 2, or 3.
```

Then they just need to write some code which beeps (see previous solution) based on that number.

# Gold.4. Sensors 📡

In this task they're instructed to detect when 2 wires are touching (effectively a button).

There's a lot of spiel in this task to try to teach them about floating pins. Ultimately the final result should be that they have the Ruggeduino (the board with 2 circuit boards and a bunch of screw terminals attached) with 2 wires, one from the 0v or GND screw terminal, the other to one of the digital IO pins (ones labelled up to 13).

Then their code should look similar to as follows:

```
from sr.robot import *
R = Robot()
their_io_pin = 13
R.ruggeduinos[0].pin_mode(their_io_pin, INPUT_PULLUP)
while True:
    if R.ruggeduinos[0].digital_read(their_io_pin):
        R.power.beep(100, note='d')
```

# Platform.1. State machines 🎰

The purpose of this task is to teach the students to not use the control flow to keep track of the state of the program, a common issue we see with their code!

They're given 3 functions, task_a(), task_b(), and task_c(), which randomly returns 0 or 1. They're told to write some code which then executes A, B, or C based on their results.

They're told there's a cool trick to do this easily, but aren't told what the trick is.

**The neat trick:**
Have a variable named `current_state`
Which can be A, B, or C.

If they look up for the challenge, you can just tell them the above and see if they can figure it out. Otherwise, tell them that you should have a single variable which stores which 'state' you're in, then use a single big loop which always checks what your code *should* be doing, and does it.

***Here's the working code:***

```
current_state = 'A'

while True:
	if current_state == 'A':
		if task_a() == 1:
			current_state = 'B'
		else:
			current_state = 'C'
	if current_state == 'B':
		if task_b() == 1:
			current_state = 'C'
		else:
			current_state = 'A'
	if current_state == 'C':
		if task_c() == 1:
			current_state = 'A'
		else:
			current_state = 'B'
```

Tell the students that this style works well with your robot code! For example, they'll have tasks like "search for cube", which could go to either "move to a different place" or "pick up cube", depending on if they found the cube, and having a single variable makes life much easier.

# Platform.2. Advanced Simulator ⚡🤖⚡

This is a very open-ended task, slowly stepping through more and more advanced code until you get a functioning robot in the simulator.

Here are the steps the robot needs to do, and some quick references to where they should look in the docs for how to do it:

1. Make it drive forward
   Relevant docs section: (studentrobotics.org/docs)
   **Programming/sr/Motors (entire page)**

2. Make it see cubes
   Relevant docs section: (studentrobotics.org/docs)
   **Programming/sr/Vision (entire page, but especially 'MARKERINFO')**

3. Make it see the closest cube
   Relevant docs section: (studentrobotics.org/docs)
   **Programming/sr/Vision (entire page, but especially 'dist')**

4. Make it drive to the cube
   Relevant docs section: (studentrobotics.org/docs)
   **Programming/sr/Motors (entire page)**

5. Make it pick up the cube (use `` `.grab()` ``, check the simulator download page on the docs)
   Relevant docs section: (studentrobotics.org/docs)
   **Programming/Simulator ('Interface' section)**

6. Make it find home (hint: the starting zone has a wall marker with the code 0, assuming you start in zone 0)
   Relevant docs section: (studentrobotics.org/docs)
   **Programming/sr/Vision (entire page, but especially '.info.code')**

7. Make it drive to the scoring zone and drop the cube.
   Relevant docs section: (studentrobotics.org/docs)
   **Programming/sr/Motors (entire page)**
   **Programming/sr/Vision (entire page, but especially '.info.code')**
   **Programming/Simulator ('interface' section)**

# Platform.3. Listen for the competition 👂

This one's super tricky, make sure the team is competent or has lots of help here, as they'll be programming in C++ and flashing the arduino.

Relevant docs section: (studentrobotics.org/docs)
**Programming/sr/Ruggeduinos/Custom Firmware (entire page)**

They should get the arduino IDE installed, copy the arduino firmware from https://github.com/srobo/ruggeduino-fw/blob/master/src/ruggeduino.cpp and then modify the code to interface with the ultrasound sensor.

They literally need to implement the protocol to interface with the ultrasound sensor, so they need to send a 10 microsecond pulse down Trig and measure the length of the pulse in the Echo pin which comes in response.

Some Arduino functions that they might find useful for implementing this are:

- `pinMode`
- `digitalWrite`
- `pulseIn`
- `delayMicroseconds`