Structured Debugging Strategies with VEX V5

Jimmy Lin, PhD
Director of Computer Science Education
VEX Robotics

Workshop Goals

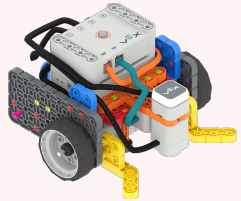
- Learn about troubleshooting techniques and debugging strategies
- Practice debugging V5 projects together
- Create your own buggy projects
- **Connect what our experience to how you approach debugging with your students**

The VEX Continuum

VEX 123

Coding Starts Early

Ages 4+



VEX GO

STEM Starts Early

Ages 8+

VEX AIM

Real World Coding

Ages 8+

VEX IQ

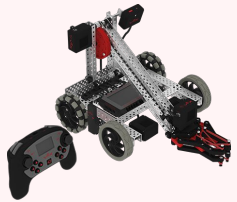
Applied STEM Learning

Ages 11+

VEX EXP

Real World STEM for Classrooms

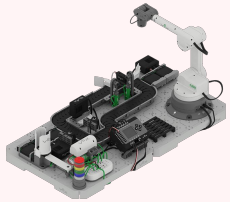
Ages 14+



VEX V5

Real World STEM for Competition

Ages 14+



VEX CTE

Workforce Readiness

Ages 14+

VEX AIR

STEM Skills Take Flight

Ages 14+

VEX CODE VR

Virtual Robot Coding

Ages 8+

A paradigm shift…

“ The idea of designing bugs for learning

—or *debugging by design*—

makes learners agents of their own learning and, more importantly, of making and solving mistakes.”

-Yasmin B. Kafai

A paradigm shift...

"...bugs not as

accidents to be solved

but as objects-to-think-with

and objects-to-share-with."

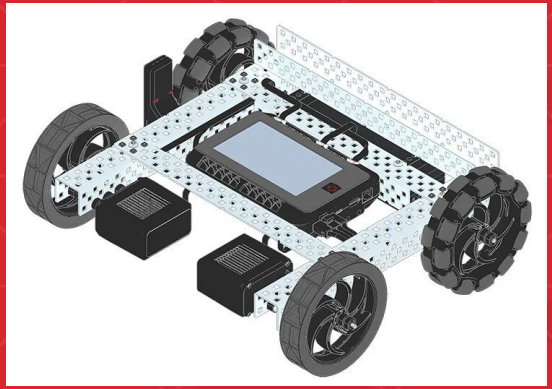
Fields, Deborah A., et al. "Debugging by design: A constructionist approach to high school students' crafting and coding of electronic textiles as failure artefacts." British Journal of Educational Technology 52.3 (2021): 1078-1092.

"Mischievousness and fun as well as empathy and sensitivity…were ***productive emotions*** exhibited during bug design, and ***shifts away from frustration to increased comfort, security, and a sense of control with bugs*** were expressed retrospectively weeks afterward."

Fields, Deborah A., et al. "Debugging by design: A constructionist approach to high school students' crafting and coding of electronic textiles as failure artefacts." British Journal of Educational Technology 52.3 (2021): 1078-1092.

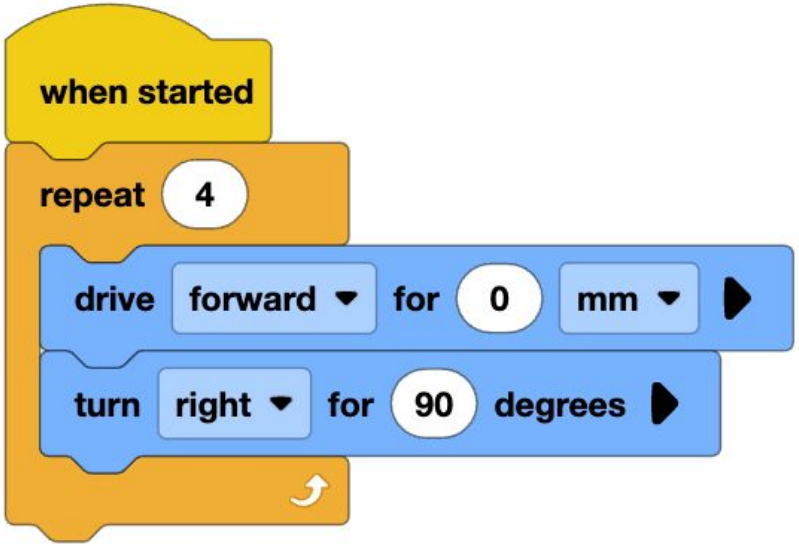
What do you need to know to debug a project?

- What is the intention of the project?
- Where does it go wrong?



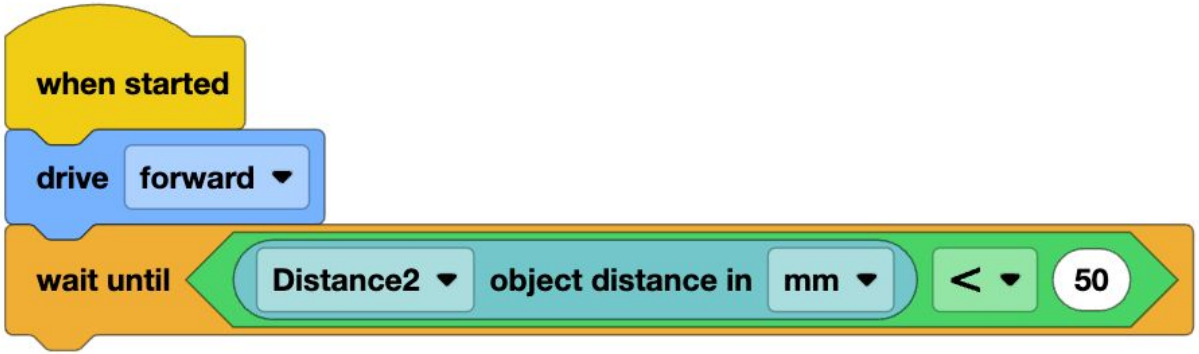
Find the Bug!

Buggy Project 1



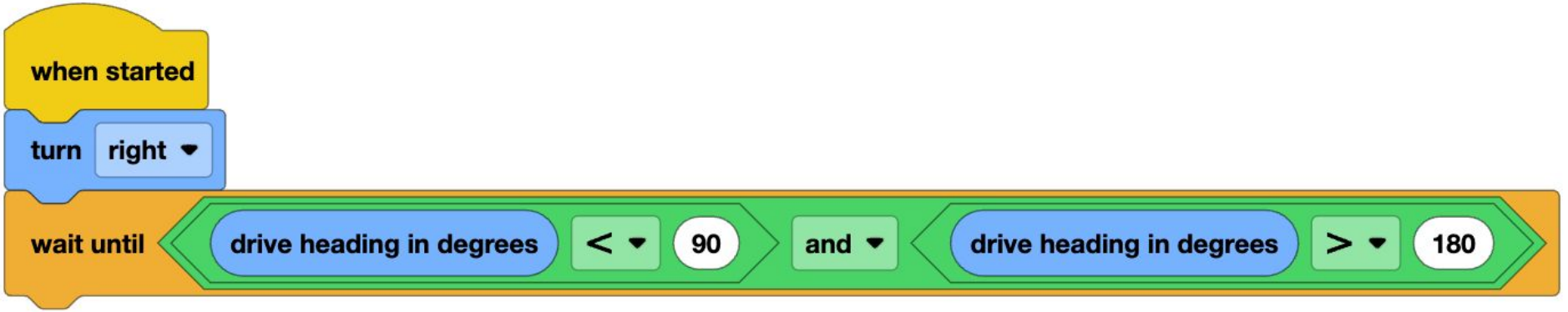

```
Drive in a square
```

Buggy Project 2



Stop until the object is close

Buggy Project 3



Turn right until
the robot is facing
the south east.

Buggy Project 4

```
50
51    # Library imports
52    from vex import *
53
54    # Drive in a square
55    for i in range(4)
56    drivetrain.drive_for(FORWARD, 200, MM)
57    drivetrain.turn_for(RIGHT, 90, DEGREES)
58
```

```
51  # Library imports
52  from vex import *
53
54  # Drive in a square
55  for i in range(4)
56  drivetrain.drive_for(FORWARD, 200, MM)
57  drivetrain.turn_for(RIGHT, 90, DEGREES)
58
59
```

Errors ⚠ 0 ✖ 1

✖ Line 55, Column 18 – expected ':' (<unknown>, line 55)

Buggy Project 4

```
55    for i in range(4):
56    drivetrain.drive_for(FORWARD, 200, MM)
57    drivetrain.turn_for(RIGHT, 90, DEGREES)
58
59
```

▼ Errors ⚠0 ✖1

✖ Line 56, Column 1 – expected an indented block after 'for' statement on line 55 (<unknown>, line 56)

Buggy Project 5

```
80  int main() {
81      // Initializing Robot Configuration. DO NOT REMOVE!
82      vexcodeInit();
83      // Begin project code
84      // Begin driving the robot forward at 20% velocity
85      Drivetrain.setDriveVelocity(20, percent)
86      Drivetrain.drive(FORWARD);
87
88  }
```

Bu

```
80  int main() {
81    // Initializing Robot Configuration. DO NOT REMOVE!
82    vexcodeInit();
83    // Begin project code
84    // Begin driving the robot forward at 20% velocity
85    Drivetrain.setDriveVelocity(20, percent)
86    Drivetrain.drive(FORWARD);
87
88  }
89
90
```

▼ Errors ⚠ 0 ✖ 2

✖ Line 56, Column 2 - expected an indented block after 'for' statement on line 55 (<unknown>, line 56)

✖ Line 85, Column 43 - expected ';' after expression

✖ Line 86, Column 20 - use of undeclared identifier 'FORWARD'

What did you notice?

- How was this experience for you?
- How does it feel when you find a bug?
- Can we think about bugs in the same ways we think about challenges?

Bug “Buckets”

Bug Buckets

- Physical Bugs
- Coding Bugs
  - Syntax bugs
  - Semantic Bugs

Physical Bugs

- Wrong ports
- Wrong device
- Loose connections
- Adding unneeded component

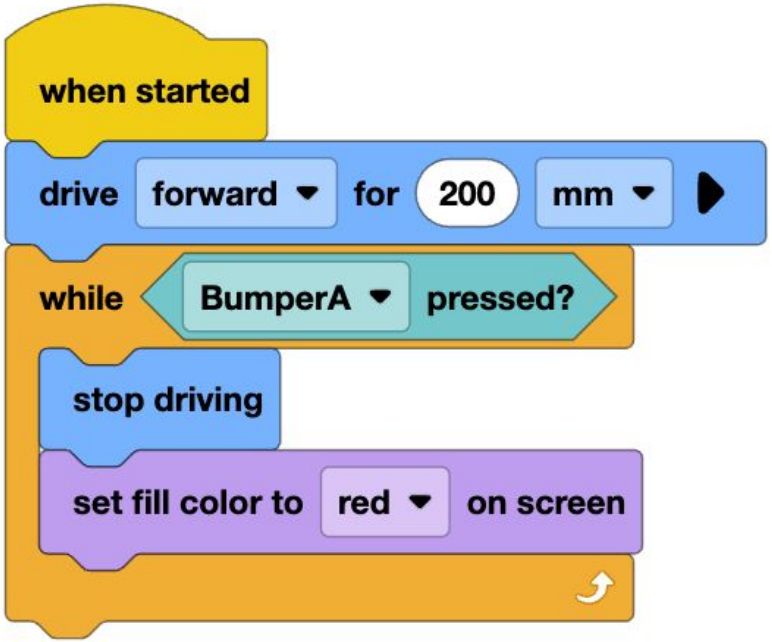
Syntax bugs

- Spaces in variable names
- Typos or mismatching variables name
- Wrong data type
- Using undeclared variables
- Calling undefined functions
- Typos in when calling function
- Missing or extra ()
- Misaligned indentation
- Case-sensitivity of constant or built-in functions

Semantic Bugs

- Missing or misplaced wait or wrong order of functions
- Reverse conditions
- Redundant logical expressions
- Contradictory logical expressions
- Gap or overlap in conditions

Your turn!



Keep driving forward until the bumper pressed and change the screen to red.

Debugging Check-in

- What were the bugs?
- How did you fix them?
- How did debugging feel for you?

Create your own bug!

Create your own bugged project to try to stump our VEXcode Experts!

- **Create a project with 2-3 bugs in it**
- Think *creatively* about the bugs you make
- Use one of the projects we've talked about or create your own

Bring this experience to your students

- **Making bugs demonstrates understanding**
  - You have to know the concept in order to create a bug with it
- **Intention is important**
  - Creative ideas are good – execution is the problem
  - Have to know what the coder was trying to do, to debug intentionally

Bring this experience to your students

- **There is more than 1 way to fix a bug**
  - Multiple solutions to the same problem helps us all learn
- **Creating bugs builds creativity and resilience**
  - The more you deal with bugs in non-threatening ways, the more persistent and comfortable you become in solving them

Stay Connected

Let's Connect!

Tag me in the **VEX PD+ Community!**
@Jimmy_Lin

Want to Learn More? Join VEX PD+ as an All-Access Member!

Schedule a **1-on-1 Session** in VEX PD+

Take a VEX Masterclass