Teacher Guide: Computer Science 2

Last Updated: January 8, 2016

Table of Contents

Summary: Computer Science 2

Armed with basic knowledge of the structure and syntax of simple programs, students are ready to tackle more advanced topics. Conditionals, arithmetic, input handling, oh my! Computer Science 2 is where students move past the programming-toy stage into writing code similar to that they would use in the next major software or killer app!

In Computer Science 2, students will continue to learn the fundamentals, (basic syntax, arguments, strings, variables, and loops) as well as being introduced to a second level of concepts for them to master. If statements allow the student to perform different actions depending on the state of the battlefield. Arithmetic will help players become more comfortable with using math in programming. All things in CodeCombat are objects, (that's the ‘object’ part of object-oriented programming,) and these things have accessible attributes, such as a Munchkin's position or a coin's value; both are important to begin visualizing the internal structure of the objects that make up their game world. Near the end of the Course there are some levels dedicated to input handling so the students can get introduced to the basic concept of events, and, well, its just great fun, too!

As with Course 1, this guide covers both the Python and Javascript solutions.

New Concepts Covered in this Course

If Statements

This is the building block of modern programming, the conditional. It's named as such because of its ability to check the conditions at the moment and perform different actions depending on the expression. The player is no longer able to assume there will be an enemy to attack, or if there is a gem to grab. Now, they need to check whether it exists, check if their abilities are ready, and check if an enemy is close enough to attack.

Arithmetic

Course 2 begins to ease the player into using math while coding. Levels catering to basic arithmetic address how to use math as needed in order to perform different actions effectively.

Input Handling

Input handling allows players to finally interact with their hero in real-time. After submitting their code, the player will be able to dynamically add flags to the battlefield to assist their hero in solving tough challenges. It helps teach simple event handling as well as being quite fun!

Level Overview & Solutions

1. Defense of Plainswood

Use your `buildXY` skills to build `"fence"`s and block out the ogres!

Remember to hover over the level map to find `x` and `y` coordinates for your `buildXY` method.
In this case, you want to build on the X marks at `40, 52` and `40, 20`.

It is much, much easier to do this level by building type `"fence"` than by building type `"fire-trap"`.

Defense of Plainswood Solution

Python

```
# Build two fences to keep the villagers safe!
# Hover your mouse over the world to get X,Y coordinates.
self.buildXY("fence", 40, 53)
self.buildXY("fence", 40, 21)
```

Javascript

```
// Build two fences to keep the villagers safe!
// Hover your mouse over the world to get X,Y coordinates.
this.buildXY("fence", 40, 53);
this.buildXY("fence", 40, 21);
```

2. Winding Trail

Forget those old simple `moveRight` boots!

Your new digs let you `moveXY` for continuous movement, wherever you want to go. They even have pathfinding built in. Sweet, huh?

Just like with `buildXY`, you can hover over the level to find `x` and `y` coordinates for you to move to.

Move to each gem in turn, then stop the ogre from getting you by building a fence on the X marker!

Winding Trail Solution

Python

```
# Go to the end of the path and build a fence there.
# Use your moveXY(x, y) function.

self.moveXY(36, 59)
self.moveXY(37, 12)
```

```
self.moveXY(66, 17)
self.buildXY("fence", 71, 24)
```

Javascript

```
// Go to the end of the path and build a fence there.
// Use your moveXY(x, y) function.

this.moveXY(36, 59);
this.moveXY(37, 12);
this.moveXY(66, 17);
this.buildXY("fence", 71, 24);
```

3. Backwoods Ambush

An **if-statement** says, **if** some condition is true, then run some code (if it's not true then don't run the code!)

To complete this level, you should move to each of the X marks with `moveXY`.

At each X spot, there may or may not be an ogre (the ogres are spawned randomly each time you press the Submit button!).

So use `findNearestEnemy` and `if` statements to determine if an ogre is at each spot, like this:

python

```
enemy = self.findNearestEnemy()
if enemy:
    self.attack(enemy)
```

javascript

```
var enemy = this.findNearestEnemy();
if(enemy) {
    this.attack(enemy);
}
```

When you use an `if` statement this way, you won't get an error by trying to attack an enemy when there is no one there!

Backwoods Ambush Solution

Python

```
# Move to each checkpoint and take out each ogre.
self.moveXY(24, 42)
enemy1 = self.findNearestEnemy()
# Use an if statement to make sure an enemy is present before attacking.
if enemy1:
      self.attack(enemy1)
      self.attack(enemy1)

self.moveXY(27, 60)
enemy2 = self.findNearestEnemy()
```

```
if enemy2:
      self.attack(enemy2)
      self.attack(enemy2)

self.moveXY(42, 50)
# Add another if statement and attack!
enemy3 = self.findNearestEnemy()
if enemy3:
    self.attack(enemy3)
    self.attack(enemy3)

# Keep moving and checking for enemies.
self.moveXY(39, 24)
enemy4 = self.findNearestEnemy()
if enemy4:
    self.attack(enemy4)
    self.attack(enemy4)

# Keep moving and checking for enemies.
self.moveXY(55, 29)
enemy5 = self.findNearestEnemy()
if enemy5:
    self.attack(enemy5)
    self.attack(enemy5)
```

Javascript

```
// Move to each checkpoint and take out each ogre.
this.moveXY(24, 42);
var enemy1 = this.findNearestEnemy();
// Use an if statement to make sure an enemy is present before attacking.
if(enemy1) {
    this.attack(enemy1);
    this.attack(enemy1);
}

this.moveXY(27, 60);
var enemy2 = this.findNearestEnemy();
if(enemy2) {
    this.attack(enemy2);
    this.attack(enemy2);
}

this.moveXY(42, 50);
// Add another if statement and attack!
var enemy3 = this.findNearestEnemy();
if(enemy3) {
    this.attack(enemy3);
    this.attack(enemy3);
}

// Keep moving and checking for enemies.
this.moveXY(39, 24);
```

```
var enemy4 = this.findNearestEnemy();
if(enemy4) {
    this.attack(enemy4);
    this.attack(enemy4);
}

// Keep moving and checking for enemies.
this.moveXY(55, 29);
var enemy5 = this.findNearestEnemy();
if(enemy5) {
    this.attack(enemy5);
    this.attack(enemy5);
}
```

4. Patrol Buster

You now have the ability to use `if-statement`s. They let you run code only if a certain condition is true.

In this level, you want to attack the nearest enemy, but only if there is an enemy. Use an **if-statement** with `enemy` as the condition to do that.

javascript
```
var enemy = this.findNearestEnemy();
if(enemy) {
    this.attack(enemy);
}
```

python
```
enemy = self.findNearestEnemy()
if enemy:
    self.attack(enemy)
```

Remember to hover over the `if/else` and read the example code in the **lower right** to see what the syntax should be.

Patrol Buster Solution

Python
```
# Remember that enemies may not yet exist.
loop:
    enemy = self.findNearestEnemy()
    # If there is an enemy, attack it!
    if enemy:
        self.attack(enemy)
```

Javascript

```
// Remember that enemies may not yet exist.
loop {
    var enemy = this.findNearestEnemy();
    // If there is an enemy, attack it!
    if(enemy) {
        this.attack(enemy);
    }
}
```

5. Endangered Burl

Each enemy has a **property** named `type`, which is a **string** (a piece of data in quotes, like `"thrower"`).

Using **if-statements** to check the `type` of an enemy allows you to decide to do different things when you see different types of enemies!

In this level, you want to `attack` enemies of type `"thrower"` and `"munchkin"`.
You should ignore enemies of `type` `"burl"`,
and run away from enemies of `type` `"ogre"`.

You can check the enemy's `type` like this:

python

```
enemy = self.findNearestEnemy()
if enemy.type is "munchkin":
    self.attack(enemy)
```

javascript

```
var enemy = this.findNearestEnemy();
if(enemy.type == "munchkin") {
    this.attack(enemy);
}
```

Note that `type` is a **property**, NOT a **method** like `moveXY(20, 20)`. Do not include `()` after `type`.

Be careful to get the syntax of the if-statements correct! Hover over the `if/else` in the lower right to see examples.

Endangered Burl Solution

Python

```
# Only attack enemies of type "munchkin" and "thrower".
# Don't attack a "burl". Run away from an "ogre"!
```

```
loop:
    enemy = self.findNearestEnemy()

    # Remember: don't attack type "burl"!
    if enemy.type is "burl":
        self.say("I'm not attacking that Burl!")

    # The "type" property tells you what kind of creature it is.
    if enemy.type is "munchkin":
        self.attack(enemy)

    # Use "if" to attack a "thrower".
    if enemy.type is "thrower":
        self.attack(enemy)
    # If it's an "ogre", run away to the village gate!
    if enemy.type is "ogre":
        self.moveXY(41, 47)
```

Javascript

```
// Only attack enemies of type "munchkin" and "thrower".
// Don't attack a "burl". Run away from an "ogre"!
loop {
    var enemy = this.findNearestEnemy();

    // Remember: don't attack type "burl"!
    if(enemy.type == "burl") {
        this.say("I'm not attacking that Burl!");
    }

    // The "type" property tells you what kind of creature it is.
    if(enemy.type == "munchkin") {
        this.attack(enemy);
    }
    // Use "if" to attack a "thrower".
    if(enemy.type == "thrower") {
        this.attack(enemy);
    }

    // If it's an "ogre", run away to the village gate!
    if(enemy.type == "ogre") {
        this.moveXY(41, 47);
    }
}
```

6. Thumb Biter

This level introduces many new things.

In order to succeed, you need to fix the `if` statements so that your hero says things to the ogre to trick him into the mines.

The block of code the `if` controls (its body) will only run if the condition (the mathy bit) works out to be True.

Enough working `if`s and the ogre will blunder into the mines trying to get at you!

If that was confusing, read on! There's more detail below:

 # Boolean
A **boolean** value means that something is either `True` or `False`.

Whether or not something is considered `True` or `False` is a complicated subject in programming, but for now we will start you off with a simple example.

 # Comparison: Equals
Use **comparison operators** to compare two values. The result of a comparison will be either True or False.

The first comparison operator we'll use is the **equality operator**. In Python and JavaScript, this is written as: `==`.

Note that this is **two equal-signs together** `==`, as opposed to `=` which is the **assignment operator** used to assign a value to a variable! *Confusing these two is a common mistake by new programmers!*

We use `==` like this:

`4 == 4` (this is **True**)
`4 == 5` (this is **False**)

We can also combine this with other mathematical operators like `+`:

`2 + 2 == 4` (this is **True**)
`2 + 2 == 5` (this is **False**)

 # Conditional Statement: if
The `if` statement says: "**if** *this* is True, **then** do *that*"

python
```
if 2 + 2 == 4:
    self.say("2 + 2 equals 4!") # Happens all the time, because 2 + 2 is 4!
if 2 + 3 == 4:
    self.say("2 + 3 equals 4!") # Will never happen, because 2 + 3 isn't 4!
```

javascript

```
if(2 + 2 == 4) {
    this.say("2 + 2 equals 4!"); // Happens all the time, because 2 + 2 is 4!
}
if(2 + 3 == 4) {
    this.say("2 + 3 equals 4!"); // Will never happen, because 2 + 3 isn't 4!
}
```

Thumb Biter Solution

Python

```
# The commands below an if statement only run when the ifâ€™s condition is true.
# In a condition, == means "is equal to."
if 2 + 2 == 4:
    self.say("Hey!")
if 2 + 2 == 5:
    self.say("Yes, you!")

# Change the condition here to make your hero say "Come at me!"
if 3 + 3 == 6:  # â^† Make this true.
    self.say("Come at me!")

if 20 == 20:  # â^† Make this true.
    # Add one more taunt to lure the ogre. Be creative!
    self.say("I double dog dare you!")
```

Javascript

```
// The commands below an if statement only run when the ifâ€™s condition is true.
// In a condition, == means "is equal to."
if(2 + 2 == 4) {
    this.say("Hey!");
}
if(2 + 2 == 5) {
    this.say("Yes, you!");
}

// Change the condition here to make your hero say "Come at me!"
if(3 + 3 == 6) { // â^† Make this true.
    this.say("Come at me!");
}

if(20 == 20) { # â^† Make this true.
    # Add one more taunt to lure the ogre. Be creative!
    this.say("I double dog dare you!");
}
```

7. Gems or Death

This level is all about the `if` statement. As a matter of fact, you don't have to write any code at all. Your job is debugging.

All you have to do is fix the `if` statements so that the actions you want your hero to take happen and the ones you don't want don't happen.

The block of code the `if` controls (its body) will only get run if the condition (the mathy bit) works out to be true.

Let's take for example the first `if`:

python
```
if 1 + 1 + 1 == 3:
    self.moveXY(5, 15)  # Move to the first mines.
```

javascript
```
if(1 + 1 + 1 == 3) {
    this.moveXY(5, 15)  # Move to the first mines.
}
```

Since `1 + 1 + 1` does equal `3` it is true. So, off into the mines we run...

If you don't want to die, change either the `1 + 1 + 1` or the `3` so that it is no longer true (further down in the code you will see that you can also change the `==`).

Then continue with each if statement, making it true or false depending on whether or not you wish its body to happen.

Gems or Death Solution

Python
```
# The commands below an if statement only run when the if's condition is true.
# Fix all the if-statements to beat the level.

# == means "is equal to".
if 1 + 1 + 1 == 4:
    self.moveXY(5, 15)  # Move to the first mines.

if 2 + 2 == 4:
      self.moveXY(15, 40)  # Move to the first gem.

# != means "is not equal to".
if 2 + 2 != 5:
      self.moveXY(25, 15)  # Move to the second_gem.

# < means "is less than".
if 2 + 2 < 5:
```

```
    enemy = self.findNearestEnemy()
    self.attack(enemy)

if 2 < 1:
      self.moveXY(40, 55)

if not True:
      self.moveXY(50, 10)

if not False:
      self.moveXY(55, 25)
```

Javascript

```
// The commands below an if statement only run when the if's condition is true.
// Fix all the if-statements to beat the level.

// == means "is equal to".
if(1 + 1 + 1 == 4) {
    this.moveXY(5, 15); // Move to the first mines.
}

if(2 + 2 == 4) {
    this.moveXY(15, 40); // Move to the first gem.
}

// != means "is not equal to".
if(2+2 != 5) {
    this.moveXY(25, 15); // Move to the second gem.
}

// < means "is less than".
if (2 + 2 < 5) {
    var enemy = this.findNearestEnemy();
    this.attack(enemy);
}

if(2 < 1) {
    this.moveXY(40, 55);
}

if(false) {
    this.moveXY(50, 10);
}

if(true) {
    this.moveXY(55, 25);
}
```

8. Village Guard

You can do this level with two **if-statements**.

The first one, with `leftEnemy`, is in the default code as an example, so reload the sample code if you get off track.

Move to the X on the right, then define a `rightEnemy` variable with your `findNearestEnemy` method.
Then, write an **if-statement** to check if `rightEnemy` exists. If there is an enemy, attack it!

Make sure that you define the `rightEnemy` variable when you would be able to see an enemy coming on the right.

Village Guard Solution

Python

```
# Patrol the village entrances.
# If you find an enemy, attack it.
loop:
    self.moveXY(35, 34)
    leftEnemy = self.findNearestEnemy()
    if leftEnemy:
        self.attack(leftEnemy)
        self.attack(leftEnemy)
    # Now move to the right entrance.
    self.moveXY(60, 31)
    # Use "if" to attack if there is an enemy.
    rightEnemy = self.findNearestEnemy()
    if rightEnemy:
        self.attack(rightEnemy)
        self.attack(rightEnemy)
```

Javascript

```
// Patrol the village entrances.
// If you find an enemy, attack it.
loop {
    this.moveXY(35, 34);
    var leftEnemy = this.findNearestEnemy();
    if(leftEnemy) {
        this.attack(leftEnemy);
        this.attack(leftEnemy);
    }

    // Now move to the right entrance.
    this.moveXY(60, 31);
    // Use "if" to attack if there is an enemy.
    var rightEnemy = this.findNearestEnemy();
    if(rightEnemy) {
        this.attack(rightEnemy);
        this.attack(rightEnemy);
    }
```

```
}
```

9. Thornbush Farm

Ogres are coming from the top, left, and bottom, so you need three sets of commands in your **loop**:
one for `topEnemy`, one for `leftEnemy`, and one for `bottomEnemy`.

Write the `leftEnemy` and `bottomEnemy` code based on the `topEnemy` sample code.

Make sure that in each set of commands, you:

1. First, `moveXY` to the X marker
2. Define a new enemy variable with `findNearestEnemy` **after** you get to the marker
3. Write an if statement: *if* there is an enemy, *then* build a `"fire-trap"` on the X marker

After that, your loop will repeat to patrol all three entrances several times.

You only want to build a fire trap if you see an ogre coming, because otherwise a peasant will try to get into the village only to be blown to smithereens by your fire trap!

If you are getting stuck, look very closely at the `topEnemy` part to make sure your code is formatted the same way for `leftEnemy` and `bottomEnemy`.

Thornbush Farm Solution

Python

```
# Patrol the village entrances.
# Build a "fire-trap" when you see an ogre.
# Don't blow up any peasants!

loop:
    self.moveXY(43, 50)
    topEnemy = self.findNearestEnemy()
    if topEnemy:
        self.buildXY("fire-trap", 43, 50)

    self.moveXY(25, 34)
    leftEnemy = self.findNearestEnemy()
    if leftEnemy:
        self.buildXY("fire-trap", 25, 34)

    self.moveXY(43, 20)
    bottomEnemy = self.findNearestEnemy()
    if bottomEnemy:
        self.buildXY("fire-trap", 43, 20)
```

Javascript

```
// Patrol the village entrances.
// Build a "fire-trap" when you see an ogre.
// Don't blow up any peasants!

loop {
    this.moveXY(43, 50);
    var topEnemy = this.findNearestEnemy();
    if(topEnemy) {
        this.buildXY("fire-trap", 43, 50);
    }

    this.moveXY(25, 34);
    var leftEnemy = this.findNearestEnemy();
    if(leftEnemy) {
        this.buildXY("fire-trap", 25, 34);
    }

    this.moveXY(43, 20);
    var bottomEnemy = this.findNearestEnemy();
    if(bottomEnemy) {
        this.buildXY("fire-trap", 43, 20);
    }
}
```

10. Back to Back

This level introduces the `else` part of `if/else`.

When you add an `else` clause, you choose what to do both when the condition is true and when it is not true.

So you can say, *if* there is an enemy, *then* attack it, *else* move to the X.

To show you how it works, the `if` and the `else` are set up for you, and you need to put in the `attack` and `moveXY` methods so that your hero attacks enemies on sight, but when there are no enemies, moves back to the X marker to defend the peasants.

Make sure you get the coordinates for the X marker correct, or you might not be able to defend both your peasants in time.

Back to Back Solution

Python

```
# Stay in the middle and defend!
while True:
    enemy = self.findNearestEnemy()
    # Either attack the enemy...
    if enemy:
        self.attack(enemy)
        self.attack(enemy)
    else:
        # ... or move back to your defensive position.
        self.moveXY(40, 34)
```

Javascript

```
// Stay in the middle and defend!
while(true) {
    var enemy = this.findNearestEnemy();
    // Either attack the enemy...
    if(enemy) {
        this.attack(enemy);
        this.attack(enemy);
    }
    // ... or move back to your defense position
    else {
        this.moveXY(40, 34);
    }
}
```

11. Ogre Encampment

For this level, you'll need to use both `if` and `else`. Remember that the `else` block executes when the `if` condition is not true.

When the ogres attack you, you want to fight back, but when there are no ogres, you can keep attacking the `"Chest"` to open it.
So in your `if` condition, check whether there is an enemy. If there is, attack it. Else, attack the `"Chest"`.

To remember the if/else syntax, either hover over the `if/else` examples in the lower right, from your Programmaticon II.

Ogre Encampment Solution

Python

```
# If there is an enemy, attack it.
# Otherwise, attack the chest!

loop:
    # Use if/else.
    enemy = self.findNearestEnemy()
    if enemy:
        self.attack(enemy)
    else:
```

```
        self.attack("Chest")
```

Javascript

```
// if there is an enemy, attack it.
// Otherwise, attack the chest!

loop {
    // Use if/else.
    var enemy = this.findNearestEnemy();
    if(enemy) {
        this.attack(enemy);
    } else {
        this.attack("Chest");
    }
}
```

12. Woodland Cleaver

The woods are swarming with ogre munchkins, but you have a new Long Sword, and its `cleave` ability will make short work of them! `cleave` hits every enemy within ten meters of your hero.

Special abilities like `cleave` have cooldown periods, which means you can't use them all the time. (You can only cleave every ten seconds.) You need to check if they are ready to use first. Fortunately, your Sundial Wristwatch gives you the `isReady` method. It tells you whether special abilities are ready to be used yet.

Putting everything together, your code should go like this:

* loop
  * find an enemy
  * *if* `"cleave"` is ready, *then*
    * cleave the enemy
  * *else*
    * attack the enemy

Hover over the `isReady` and `cleave` documentation in the lower right to see the syntax for how to use them.

Woodland Cleaver Solution

Python

```
# Use your new "cleave" skill as often as you can.

self.moveXY(23, 23)
loop:
    enemy = self.findNearestEnemy()
    if self.isReady("cleave"):
        # Cleave the enemy!
        self.cleave(enemy)
    else:
```

```
        # Else (if cleave isn't ready), do your normal attack.
        self.attack(enemy)
```

Javascript

```
// Use your new "cleave" skill as often as you can.

this.moveXY(23, 23);
loop {
    var enemy = this.findNearestEnemy();
    if(this.isReady("cleave")) {
        // Cleave the enemy!
        this.cleave(enemy);
    } else {
        // Else (if cleave isn't ready), do your normal attack.
        this.attack(enemy);
    }
}
```

13. Shield Rush

Your shield gives you the `shield()` ability, which blocks some of the damage you take while you are shielding.
You can't do anything else while shielding, but it's a useful ability to help you stay alive until you can use `cleave()` again.

Remember that special abilities like `cleave` have cooldown periods, which means you can't use them all the time.
(You can only cleave every ten seconds.) You need to check if they are ready to use with your Sundial Wristwatch's `isReady()` method.

You'll need to use `if/else` like this:

javascript

```
while(true) {
    var enemy = this.findNearestEnemy();
    if(this.isReady("cleave")) {
        // Cleave!
    } else {
        // Shield yourself!
    }
}
```

python

```
while True:
    enemy = self.findNearestEnemy()
    if self.isReady("cleave")
        # Cleave!

    else
        # Shield!
```

Hover over the `isReady`, `cleave`, `shield`, and `if/else` documentation in the lower right if you need a reminder on the syntax.

Tip: **use a loop**! The sample code won't always give you the `while` statement.
You need a loop so that you can decide what to do over and over instead of just once in the beginning.

Shield Rush Solution

Python

```
# Survive both waves by shielding and cleaving.
# When "cleave" is not ready, use your shield skill.
# You'll need at least 142 health to survive.
loop:
    enemy = self.findNearestEnemy()
    if self.isReady("cleave"):
        self.cleave(enemy)
    else
        self.shield()
```

Javascript

```
// Survive both waves by shielding and cleaving.
// When "cleave" is not ready, use your shield skill.
// You'll need at least 142 health to survive.
loop {
    var enemy = this.findNearestEnemy();
    if(this.isReady("cleave") {
        this.cleave(enemy);
    } else {
        this.shield();
    }
}
```

14. Range Finder

You've been asked to test special glasses that can see through trees! This time, you don't need to go out and deal with the ogres personally.

Your artillery can't sight through the trees, so use `distanceTo()` and `say()` to call out the range to each target.

Be careful, though! There are peaceful woodsmen living in these woods.

Range Finder Solution

Python

```
enemy1 = "Gort"
enemy2 = "Smasher"
enemy3 = "Charles"
enemy4 = "Gorgnub"

distance1 = self.distanceTo(enemy1)
self.say(distance1)
# The artillery will destroy Gort!
# Find the distance to the other two ogres.
# Give the command to fire by saying the distance.
distance2 = self.distanceTo(enemy2)
self.say(distance2)
distance3 = self.distanceTo(enemy4)
self.say(distance3)
```

Javascript

```
var enemy1 = "Gort";
var enemy2 = "Smasher";
var enemy3 = "Charles";
var enemy4 = "Gorgnub";

var distance1 = this.distanceTo(enemy1);
this.say(distance1);
// The artillery will destroy Gort!
// Find the distance to the other two ogres.
// Give the command to fire by saying the distance.
var distance2 = this.distanceTo(enemy2);
this.say(distance2);
var distance3 = this.distanceTo(enemy4);
this.say(distance3);
```

15. Peasant Protection

You can use `distanceTo` to measure the distance, in meters, to a target.
In this level, you'll use that to make sure you stay close to vulnerable peasant Victor.

You can see a new piece of syntax here, the **less-than** operator: `<`

You can read it like this: *if* the distance is *less than* 10 meters, *then* attack the enemy, *else* move back to the X marker.

Fill in the `else` to move back to the X so that no ogres can get to Victor while you're far away.

Tip: make sure that you are moving to the correct defensive position – the X is at `{x: 40, y: 37}`.

Peasant Protection Solution

Python

```
loop:
    enemy = self.findNearestEnemy()
    distance = self.distanceTo(enemy)
    if distance < 10:
        # Attack if they get too close to the peasant.
        self.attack(enemy)

    # Else, stay close to the peasant! Use else.
    else:
        self.moveXY(40, 37)
```

Javascript

```
loop {
    var enemy = this.findNearestEnemy();
    var distance = this.distanceTo(enemy);
    if(distance < 10) {
        // Attack if they get too close to the peasant.
        this.attack(enemy);
    }
    // Else, stay close to the peasant! Use else.
    else {
        this.moveXY(40, 37);
    }
}
```

16. Munchkin Swarm

In this level, you put together everything you've learned over the past few levels in order to use `if/else`, `distanceTo`, `<`, and `cleave` to defeat vast numbers of ogre munchkins while looting a giant treasure chest.

These munchkins have become suitably terrified of you and your mighty Long Sword, so they will only approach when there are a lot of them together in a pack. Check the distance to the nearest munchkin and only `cleave` if the munchkin is closer than ten meters. Use an else clause to attack the `"Chest"` otherwise.

Tip: make sure to use a **while-true loop**.

Tip: you'll know that your distance check is working when your hero never chases any munchkins away from the chest.

Munchkin Swarm Solution

Python

```
loop:
    # Check the distance to the nearest enemy.
    enemy = self.findNearestEnemy()
    distance = self.distanceTo(enemy)
    # If it comes closer than 10 meters, cleave it!
    if distance < 10:
        self.cleave(enemy)
```

```
    else:
        # Else, attack the "Chest" by name.
        self.attack("Chest")
```

Javascript

```
loop {
    // Check the distance to the nearest enemy.
    var enemy = this.findNearestEnemy();
    var distance = this.distanceTo(enemy);
    //If it comes closer than 10 meters, cleave it!
    if(distance < 10) {
        this.cleave(enemy);
    } else {
        // Else, attack the "Chest" by name.
        this.attack("Chest");
    }
}
```

17. Stillness in Motion

For this level, you want to stay in the middle where the Headhunters can't see you!

You will use **nested-if-statements**.

When dealing with nested if statements, you need to pay close attention to how you set up the flow of your program.

If your if statements are complicated, it's often easier to build them up one at a time, using comments to fill in the future statements.
For example, in this level we could begin by writing the following:

python

```
# If there is an enemy, then...
    # Do something
# Otherwise (there is no enemy)...
    # Move back to the X
```

javascript

```
// If there is an enemy, then...
    // Do something
// Otherwise (there is no enemy)...
    // Move back to the X
```

Next, fill in the outer if/else statements, and the move, for real:

python

```
if enemy:
```

```
    # Do something
else:
    self.moveXY(40, 34)
```

javascript

```
if(enemy) {
    // Do something
 } else {
    this.moveXY(40, 34)
}
```

Now, let's detail the "Do something" :

python

```
if enemy:
    # If the enemy is less than 5 meters away, then attack
    # Otherwise (the enemy is far away), shield()
else:
    self.moveXY(40, 34)
```

javascript

```
if(enemy) {
    // If the enemy is less than 5 meters away, then attack
    // Otherwise (the enemy is far away), shield()
 } else {
    this.moveXY(40, 34)
}
```

Finally, fill in the actual code for the inner if/else, making sure everything is indented correctly:

python

```
if enemy:
    if self.distanceTo(enemy) < 5:
        self.attack(enemy)
    else:
        self.shield()
else:
    self.moveXY(40, 34)
```

javascript

```
if(enemy) {
    if(this.distanceTo(enemy) < 5) {
        this.attack(enemy);
```

```
    } else {
        this.shield();
    }
 } else {
    this.moveXY(40, 34)
}
```

And, this entire block of if-s and else-s has to be indented under the **while-true loop** like:

```
python
while True:
    enemy = self.findNearestEnemy()

    if enemy:
        if self.distanceTo(enemy) < 5:
            self.attack(enemy)
        else:
            self.shield()
    else:
        self.moveXY(40, 34)

javascript
while(true) {
    var enemy = self.findNearestEnemy();

    if(enemy) {
        if(this.distanceTo(enemy) < 5) {
            this.attack(enemy);
        } else {
            this.shield();
        }
     } else {
        this.moveXY(40, 34)
    }

}
```

Hint: You can highlight several lines of code and press *Tab* to indent all of those lines, or *Shift+Tab* to un-indent all of those lines!

Stillness in Motion Solution

Python

```
# You can put one if-statement within another if-statement.
# However, doing so can be tricky, so you have to pay close attention to how the if
statements interact with each other.
# Make sure the indentation is correct!
# Use comments to describe your logic in plain language!
# It's helpful to start with one outer if/else, using comments as placeholders for the inner
if/else, like so:
loop:
    enemy = self.findNearestEnemy()
    # If there is an enemy, then...
    if enemy:
        # If the enemy is less than 5 meters away, then attack
        if self.distanceTo(enemy) < 5:
            self.attack(enemy)
        # Otherwise (the enemy is far away), shield()
        else:
            self.shield()
    # Otherwise (there is no enemy)...
    else:
        # Move back to the X
        self.moveXY(40, 34)
```

Javascript

```
// You can put one if-statement within another if-statement.
// However, doing so can be tricky, so you have to pay close attention to how the if
statements interact with each other.
// Make sure the indentation is correct!
// Use comments to describe your logic in plain language!
// It's helpful to start with one outer if/else, using comments as placeholders for the
inner if/else, like so:
loop {
    var enemy = this.findNearestEnemy();

    // If there is an enemy, then...
    if(enemy) {
        // If the enemy is less than 5 meters away, then attack
        if(this.distanceTo(enemy) < 5) {
            this.attack(enemy);
        }
        // Otherwise (the enemy is far away), shield()
        else {
            this.shield();
        }
    }
    // Otherwise (there is no enemy)...
    else {
        // Move back to the X.
        this.moveXY(40, 34);
    }
}
```

18. The Agrippa Defense

Sometimes it's best not to open with your strongest attack immediately. If you cleave at the first sight of the enemy, you may only catch the first few, leaving their friends to finish you off!

Try using `distanceTo()` to wait until the enemy is closer before you cleave. You can experiment to find the best range at which to strike; in this level, around **5 meters** works well.

Hint: If your cleave isn't ready, don't just stand there! Use a normal `attack()` while you wait for it to be ready again.

The Agrippa Defense Solution

Python

```
loop:
    enemy = self.findNearestEnemy()
    if enemy:
        # Find the distance to the enemy with distanceTo.
        distance = self.distanceTo(enemy)
        # If the distance is less than 5 meters...
        if distance < 5:
            # ... if "cleave" is ready, cleave!
            if self.isReady("cleave"):
                self.cleave(enemy)
            # ... else, just attack.
            else:
                self.attack(enemy)
```

Javascript

```
loop {
    var enemy = this.findNearestEnemy();
    if(enemy) {
        // Find the distance to the enemy with distanceTo.
        var  = this.distanceTo(enemy);
        // If the distance is less than 5 meters...
        if(distance < 5) {
            // ... if "cleave" is ready, cleave!
            if(this.isReady("cleave")) {
                this.cleave(enemy);
            }
            // ... else, just attack.
            else {
                this.attack(enemy);
            }
        }
    }
}
```

19. Coinucopia

Now that you have basic flags, you can submit your code to try to beat the level in real-time. As the level is running, you can control your hero by placing flags that your code can respond to.

Read the sample code in this level to understand how flags work, then press Submit and start placing flags where the coins are. You'll have to be quick to get 20 gold in 40 seconds.

The flag buttons will show up in the lower left after you press Submit.

Copper coins are worth 1 gold, silver coins are worth 2 gold, and gold coins are worth 3 gold.

Tip: you don't need to change the sample code to beat this level, just place flags after hitting Submit.

Coinucopia Solution

Python

```
# Press Submit when you are ready to place flags.
# Flag buttons appear in the lower left after pressing Submit.
while True:
    flag = self.findFlag()
    if flag:
        self.pickUpFlag(flag)
    else:
        self.say("Place a flag for me to go to.")
```

Javascript

```
// Press Submit when you are ready to place flags.
// Flag buttons appear in th elower left after pressing Submit.
while(true) {
    var flag = this.findFlag();
    if(flag) {
        this.pickUpFlag(flag);
    } else {
        this.say("Place a flag for me to go to.");
    }
}
```

20. Copper Meadows

Use your `pickUpFlag` method to go to and pick up flags that you place.
Your new glasses have the findNearestItem method, which lets your hero automatically pick up coins, but only when in line of sight.
Use flags to guide your hero to each meadow full of coins.

You can move to an item's position like this:

```
javascript
var item = this.findNearestItem();
if (item) {
    var position = item.pos;
    var x = position.x;
    var y = position.y;
    this.moveXY(x, y);
}

python
item = self.findNearestItem()
if item:
    position = item.pos
    x = position.x
    y = position.y
    self.moveXY(x, y)
```

Each item is an **object**, which is a type of data, like a **string** or a **number**. Objects contain other pieces of data, known as **properties**.

Each item object (and each unit) has a `pos` property, which stands for its position.
And each `pos` is itself an object, which has `x` and `y` properties that you can use with `moveXY` and `buildXY`.

Tip: remember that you need to press Submit before you can place flags. The meadows are randomized, so the layout will change each time.

Copper Meadows Solution

Python

```
# Collect all the coins in each meadow.
# Use flags to move between meadows.
# Press Submit when you are ready to place flags.

loop:
    flag = self.findFlag()
    if flag:
        pass  # pass is a placeholder, it has no effect.
        # Pick up the flag.
        self.pickUpFlag(flag)
    else:
        # Automatically move to the nearest item you see.
        item = self.findNearestItem()
        if item:
            position = item.pos
            x = position.x
            y = position.y
            self.moveXY(x, y)
```

Javascript

```
// Collect all the coins in each meadow.
// Use flags to move between meadows.
// Press Submit when you are ready to place flags.

loop {
    var flag = this.findFlag();
    if(flag) {
        // Pick up the flag.
        this.pickUpFlag(flag);
    } else {
        var item = this.findNearestItem();
        if(item) {
            var position = item.pos;
            var x = position.x;
            var y = position.y;
            this.moveXY(x, y);
        }
    }
}
```

21. Drop the Flag

Use your `pickUpFlag` method to go to and pick up flags that you place – but first, use `buildXY` to build a `"fire-trap"` where the flag is.

Just like in the last level, where each coin item is an object, each flag is also an object. Each flag and item object has a `pos` property, which stands for its position. And each `pos` is itself an object, which has `x` and `y` properties that you can use with `moveXY` and `buildXY`.

Code your hero to build traps where she sees flags, and then when you see an ogre coming, place a flag on the X so your hero responds. When there is no flag, your hero will collect coins. **Wait for your hero to pick up the flag** before placing another one, or she won't place the fire-trap at the second flag.

Tip: remember that you need to press Submit before you can place flags. The ogres are randomized, so they'll come from different paths each time.

Drop the Flag Solution

Python

```
# Put flags where you want to build traps.
# When you're not building traps, pick up coins!

loop:
    flag = self.findFlag()
    if flag:
        # How do we get fx and fy from the flag's pos?
        # (Look below at how to get x and y from items.)
```

```
        flagPos = flag.pos
        flagX = flagPos.x
        flagY = flagPos.y

        self.buildXY("fire-trap", fx, fy)
        self.pickUpFlag(flag)
    else:
        item = self.findNearestItem()
        if item:
            itemPos = item.pos
            itemX = itemPos.x
            itemY = itemPos.y
            self.moveXY(itemX, itemY)
```

Javascript

```
// Put flags where you want to build traps.
// When you're not building traps, pick up coins!
loop {
    var flag = this.findFlag();
    if(flag) {
        // How do we get fx and fy from the flag's pos?
        // (Look below at how to get x and y from items.)
        var flagPos = flag.pos;
        var flagX = flagPos.x;
        var flagY = flagPos.y;

        this.buildXY("fire-trap", flagX, flagY);
        this.pickUpFlag(flag);
    } else {
        var item = this.findNearestItem();
        if(item) {
            var itemPos = item.pos;
            var itemX = itemPos.x;
            var itemY = itemPos.y;
            this.moveXY(itemX, itemY);
        }
    }
}
```

22. Mind the Trap

Some actions your hero takes will pause the rest of your program while they happen. One of these is `attack`.

When you do an `attack` against an enemy that's far away, your program will stop responding to other commands (like `pickUpFlag`) while your hero runs toward the enemy.

In this level, that means your hero will run straight into the mines! (**boom!**)

To avoid this, you'll use `distanceTo`, and only attack enemies if they are within `10` meters of you.

Then, use your flags to move close to any enemy you want to attack!

Mind the Trap Solution

Python

```
# If you try to attack a distant enemy, your hero will charge toward it, ignoring all flags.
# You'll need to make sure you only attack enemies who are close to you!

loop:
    flag = self.findFlag()
    enemy = self.findNearestEnemy()

    if flag:
        # Pick up the flag.
        self.pickUpFlag(flag)
    elif enemy:
        # Only attack if the enemy distance is < 10 meters
        if self.distanceTo(enemy) < 10:
            self.attack(enemy)
```

Javascript

```
// If you try to attack a distant enemy, your hero will charge toward it, ignoring all
flags.
// You'll need to make sure you only attack enemies who are close to you!

loop {
    var flag = this.findFlag();
    var enemy = this.findNearestFlag();
    if(flag) {
        // Pick up the flag.
        this.pickUpFlag(flag);
    }
    else if (enemy) {
        // Only attack if the enemy distance is < 10 meters
        if(this.distanceTo(enemy) < 10) {
            this.attack(enemy);
        }
    }
}
```

23. Signal Corpse

Previously, you used `distanceTo` to attack only nearby enemies, and you used flags to move closer.

Now, we'll do the same thing, but we use a `"green"` flag to move toward (or run away from!) enemies and the `"black"` flag to tell our hero to use a `'cleave'` attack.

This way, we can save the `cleave` attack for the right moment, when there are many enemies nearby.

Tip: use `cleave` with no arguments to cleave where you're standing, instead of chasing an enemy to cleave.

Signal Corpse Solution

Python

```
# You can use flags to choose different tactics.
# In this level, the green flag will mean you want to move to the flag.
# The black flag means you want to cleave at the flag.
# The doctor will heal you at the Red X

loop:
    green = self.findFlag("green")
    black = self.findFlag("black")
    nearest = self.findNearestEnemy()

    if green:
        self.pickUpFlag(green)
    elif black:
        self.pickUpFlag(black)
        # If cleave isReady, do a cleave!
        if self.isReady("cleave"):
            self.cleave()
    elif nearest:
        if self.distanceTo(nearest) < 10:
            # Attack!
            self.attack(nearest)
```

Javascript

```
// You can use flags to choose different tactics.
// In this level, the green flag will mean you want to move to the flag.
// The black flag means you want to cleave at the flag.
// The doctor will heal you at the Red X
loop {
    var green = this.findFlag("green");
    var black = this.findFlag("black");
    var nearest = this.findNearestEnemy();

    if(green) {
        this.pickUpFlag(green);
    } else if (black) {
        this.pickUpFlag(black);
        // If cleave isReady, do a cleave!
        if(this.isReady("cleave") {
            this.cleave();
        }
    } else if (nearest)
        if(this.distanceTo(nearest) < 10) {
            # Attack!
            this.attack(nearest);
        }
```

```
    }
}
```

24. Rich Forager

Combine everything you know about if/else, using flags, your special abilities, and accessing x and y coordinates from pos objects to clear all the meadows of coins and enemies.

You'll need to use `pickUpFlag` to move your hero between meadows, `attack` and `cleave` to kill enemies, and `moveXY` to move to the position of coin items that you see.

Tip: remember that you need to press Submit before you can place flags. The layouts are randomized, so they'll change each time.

Rich Forager Solution

Python

```
# Use "if" and "else if" to handle any situation.
# Put it all together to defeat enemies and pick up coins!
# Make sure you bought great armor from the item shop! 400 health recommended.

loop:
    flag = self.findFlag()
    enemy = self.findNearestEnemy()
    item = self.findNearestItem()

    if flag:
        # What happens when I find a flag?
        self.pickUpFlag(flag)
    elif enemy:
        # What happens when I find an enemy?
        if self.isReady("cleave"):
            self.cleave(enemy)
        else:
            self.attack(enemy)
    elif item:
        # What happens when I find an item?
        self.moveXY(item.pos.x, item.pos.y)
```

Javascript

```
# Use "if" and "else if" to handle any situation.
# Put it all together to defeat enemies and pick up coins!
# Make sure you bought great armor from the item shop! 400 health recommended.

loop {
    var flag = this.findFlag();
    var enemy = this.findNearestEnemy();
    var item = this.findNearestItem();
    if(flag) {
        // What happens when I find a flag?
        this.pickUpFlag(flag);
```

```
    } else if (enemy) {
        // What happens when I find an enemy?
        if(this.isReady("cleave")) {
            this.cleave(enemy);
        } else {
            this.attack(enemy);
        }
    } else if (item) {
        // What happens when I find an item?
        this.moveXY(item.pos.x, item.pos.y);
    }
}
```

25. Cross Bones

Cross Bones
Welcome to the entrance of the Sarven Desert. A valuable piece of terrain which could shifts the battle between Ogres and Humans significantly.

Collect coins to fund the effort. Summon mercenaries by stepping on the corresponding X-mark in front of the tents.

Heros
The fair `knight` **Tharin Thunderfist** is leading the charge, seeking to turn the tide of the war!

The merciless `headhunter` **Deadtooth** eager to halt humanity's advance into the desert.

Guardians
Both sides are defended by a lumbering unit which helps deter initial waves, but, even they can run out of steam. Be sure to **support** and **heal** them when you can!

The humans are protected by the not-so-gentle `goliath` **Okar Stompfoot**.

The ogres are protected by the powerful `brawler` **Grul'Thok**.

Units
Simply run over X-mark in front of the tents to summon troops.

The human side has access to the standard fare `soldier` and `archer` to recruit.
* Soldiers cost **20 gold**.
* Archers cost **25 gold**.

The ogre side has access to the desert-hardened scout and regular thrower.
* 2 Scouts cost **30 gold**.
* 2 Throwers cost **20 gold**.

Potion
Running over and collecting the potion heals both you and your guardian substantially. **Don't use it too early!**

Flags

Occasionally certain events happen which cause your guardian to toss down a flag, signalling help from your hero.

* The guardian will place a "`green`" flag when **he is under 50% health**.
* The guardian will place a "`black`" flag when **being assaulted by a large quantity of enemy units**.