Computer Science 3

JavaScript

Introduces arithmetic, counters, advanced while loops, break, continue, arrays.

#1. Friend and Foe

Level Overview and Solutions

Intro

You can combine two strings into one using the the `+` operator.

This is called concatenating strings.

```
hero.say("To battle, " + "Sally!");
```

This will cause your hero to say `"To battle, Sally!"`.

You can also concatenate strings stored in variables:

```
hero.say("To battle, " + peasant.id);
```

Default Code

```
// Peasants and peons are gathering in the forest.
// Command the peasants to battle and the peons to go away!

while(true) {
    var friend = hero.findNearestFriend();
    if(friend) {
        hero.say("To battle, " + friend.id + "!");
    }
    // Now find the nearest enemy and tell them to go away.

}
```

Overview

Just like numbers can be added together using `+`, strings also can make use of the + operator.

If you have two strings `"foo"` and `"bar"`, what do you expect the result of `"foo" + "bar"` to be?

It's very simple, in fact, as `"foo" + "bar"` is the same as `"foobar"`! It simply combines (or `concatenates`) the two strings together.

In this level you will need to command peasants to battle and peons to go away by combining their `id` with another string.

Commanding peasants has be shown to you:

```
hero.say("To battle, " + peasant.id + "!");
// This results in something like: "To battle, Sally!"
```

Please note, peons and peasants don't like it when you simply call their name! You need to add (or `concatenate`) something to the strings to get them to act.

Friend and Foe Solution

```
// Peasants and peons are gathering in the forest.
// Command the peasants to battle and the peons to go away!

while(true) {
    var friend = hero.findNearestFriend();
    if(friend) {
        hero.say("To battle, " + friend.id + "!");
    }
    // Now find the nearest enemy and tell them to go away.
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        hero.say("Go away, " + enemy.id);
    }
}
```

#2. Deja Brew

Level Overview and Solutions

Intro

Take 1 down, pass it around!
Take 1 down, pass it around!
Take 1 down, pass it around!
Take 1 down, pass it around!
Take 1 down, pass it around!
Take 1 down, pass it around!
Take 1 down, pass it around!
Take 1 down, pass it around!
Take 1 down, pass it around!

You can `concatenate` (combine) strings with other strings, or numbers:

```
var numberOfPotions = 5;
hero.say("I have " + numberOfPotions + " potions.");
```

Use string concatenation to sing along with your friends!

Default Code

```
// You can add strings together, and add numbers into strings.
// Sing along, using string concatenation:
// X potions of health on the wall!
// X potions of health!
// Take Y down, pass it around!
// X-Y potions of health on the wall.

var potionsOnTheWall = 10;
var numToTakeDown = 1;
while(true) {
    hero.say(potionsOnTheWall + " potions of health on the wall!");
    // Sing the next line:

    // Sing the next line:

    potionsOnTheWall -= numToTakeDown;
    // Sing the last line:

}
```

Overview

You can `concatenate` (combine) strings with other strings, or numbers:

```
var numberOfPotions = 5;
hero.say("I have " + numberOfPotions + " potions.");
```

Use string concatenation to sing along with your friends!

The song goes like this:

```
potionsOnTheWall + " potions of health on the wall!"
potionsOnTheWall + " potions of health!"
"Take " + numToTakeDown + " down, pass it around!"
potionsOnTheWall + " potions of health on the wall."
```

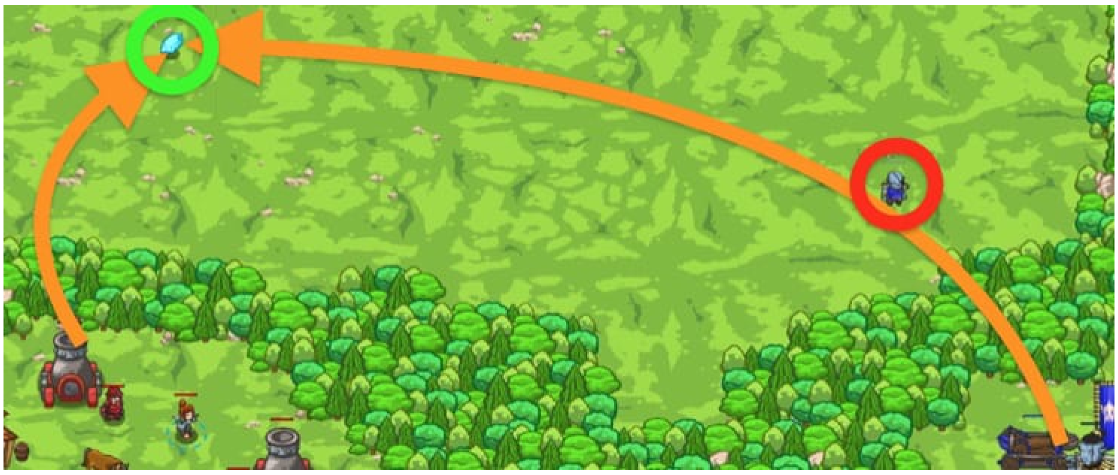
Deja Brew Solution

```
// You can add strings together, and add numbers into strings.
// Sing along, using string concatenation:
// X potions of health on the wall!
// X potions of health!
// Take Y down, pass it around!
// X-Y potions of health on the wall.

var potionsOnTheWall = 10;
var numToTakeDown = 1;
while(true) {
    hero.say(potionsOnTheWall + " potions of health on the wall!");
    // Sing the next line:
    hero.say(potionsOnTheWall + " potions of health!");
    // Sing the next line:
    hero.say("Take " + numToTakeDown + " down, pass it around!");
    potionsOnTheWall -= numToTakeDown;
    // Sing the last line:
    hero.say(potionsOnTheWall + " potions of health on the wall.");
}
```

#3. Reward and Ruination

Level Overview and Solutions

Intro



Concatenate the positions of peons and gems into easy to read strings for the artillery to shoot at.

```
var enemy = hero.findNearestEnemy();
var enemyPos = enemy.pos.x + " " + enemy.pos.y; // This is a string like: "43 20"
hero.say("Enemy at: " + enemyPos); // This says a string like: "Enemy at: 43 20"
```

Default Code

```
// It seems like the Ogre Chieftain is stealing your gems!
// Use the two artillery cannons to defeat your enemies and gather the gems.

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        var enemyPos = enemy.pos.x + " " + enemy.pos.y;
        hero.say("Enemy at " + enemyPos);
    }
    // Now that you have sweet revenge, why not have your cake and eat it, too?
    // Find the item's position and say it for your artillery to target.

}
```

Overview

As teased in the previous level, string concatenation can be done on the same line, with multiple strings.

`"Hello" + " " + "World" + "!"` is the same as `"Hello World!"`.

To beat this level you will need to concatenate the positions of peons and gems into easy to read strings for the artillery to shoot at.

```
var enemy = hero.findNearestEnemy();
var enemyPos = enemy.pos.x + " " + enemy.pos.y; // This is a string like: "43 20"
hero.say("Enemy at: " + enemyPos); // This says a string like: "Enemy at: 43 20"
```

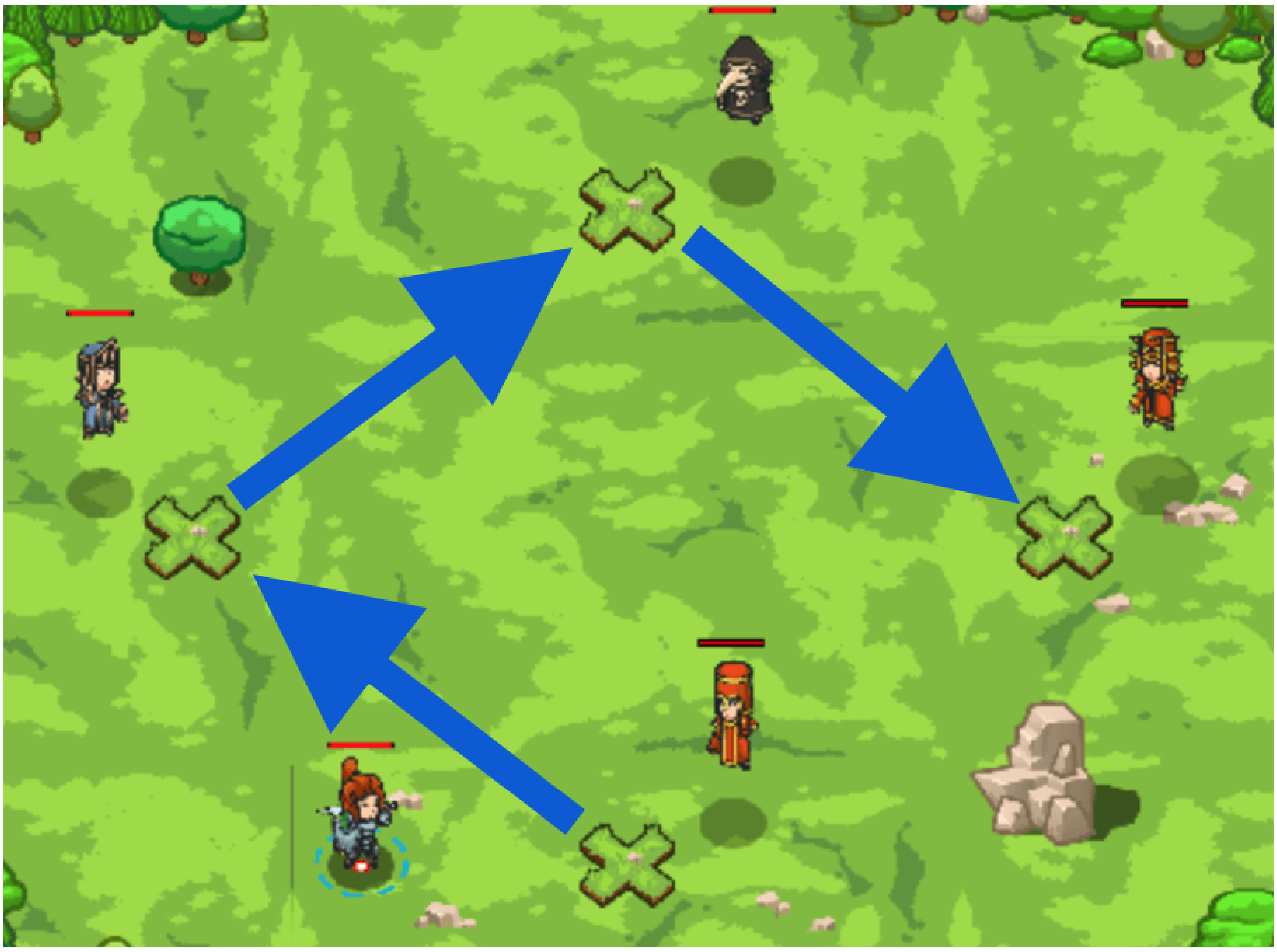
Reward and Ruination Solution

```
// It seems like the Ogre Chieftain is stealing your gems!
// Use the two artillery cannons to defeat your enemies and gather the gems.

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        var enemyPos = enemy.pos.x + " " + enemy.pos.y;
        hero.say("Enemy at " + enemyPos);
    }
    // Now that you have sweet revenge, why not have your cake and eat it, too?
    // Find the item's position and say it for your artillery to target.
    var item = hero.findNearestItem();
    if(item) {
        var itemPos = item.pos.x + " " + item.pos.y;
        hero.say("Item at " + itemPos);
    }
}
```

#4. The Wizard's Door

Level Overview and Solutions

Intro



The wizards guard a hidden door to a secret lair full of magic and treasure!

To gain access, you'll need to use math to compute magic numbers.

Listen carefully to what each wizard says to figure out how to calculate each magic number. You'll need to take things one wizard at a time.

Look at the Hints to learn how to write out math formulas in code.

Default Code

```
// Move to Laszlo and get his secret number.
hero.moveXY(30, 13);
var las = hero.findNearestFriend().getSecret();

// Add 7 to Laszlo's number to get Erzsebet's number.
// Move to Erzsebet and say her magic number.
var erz = las + 7;
hero.moveXY(17, 26);
hero.say(erz);

// Divide Erzsebet's number by 4 to get Simonyi's number.
// Go to Simonyi and tell him his number.

// Multiply Simonyi's number by Laszlo's to get Agata's number.
// Go to Agata and tell her her number.
```

Overview

In this level, you walk around to each wizard, following their instructions to calculate the magic numbers you need to speak.

You'll have to edit and run your program multiple times to get all the instructions; take it one step at a time and you'll work your way through it.

Because the secret number is randomly chosen, you can't just do the math in your head! You'll have to write code to figure out each of the secret numbers.

Doing math in code is mostly like doing it on paper, but with a few key differences. To add or subtract, you use the plus (`+`) and minus (`-`) signs:

```
var a = 4 + 5;   // a = 9
var b = 7 - 3;   // b = 4
```

But to multiply you use the asterisk (`*`) symbol:

```
var a = 2 * 3;   // a = 6
```

And to divide you use the slash (`/`) symbol:

```
var a = 6 / 2;   // a = 3
```

When you divide two numbers, there might be a remainder. You can use the modulo (`%`) operation to find out what the remainder is:

```
var a = 7 % 3;   // a = 1
var b = 9 % 5;   // b = 4
var c = 8 % 4;   // c = 0 (no remainder)
```

You've already been using *variables* to hold on to things like enemies, but you can use variables to hold anything! In the examples above, `a` , `b` , and `c` are variables that hold number values. When a variable holds a number value, you can use it in a math formula just like any other number:

```
var a = 3 * 4;   // a = 12
var b = a / 6;   // b = 12 ÷ 6 = 2
var c = b + a;   // c = 2 + 12 = 14
```

Using variables in math calculations is super important for solving this level.

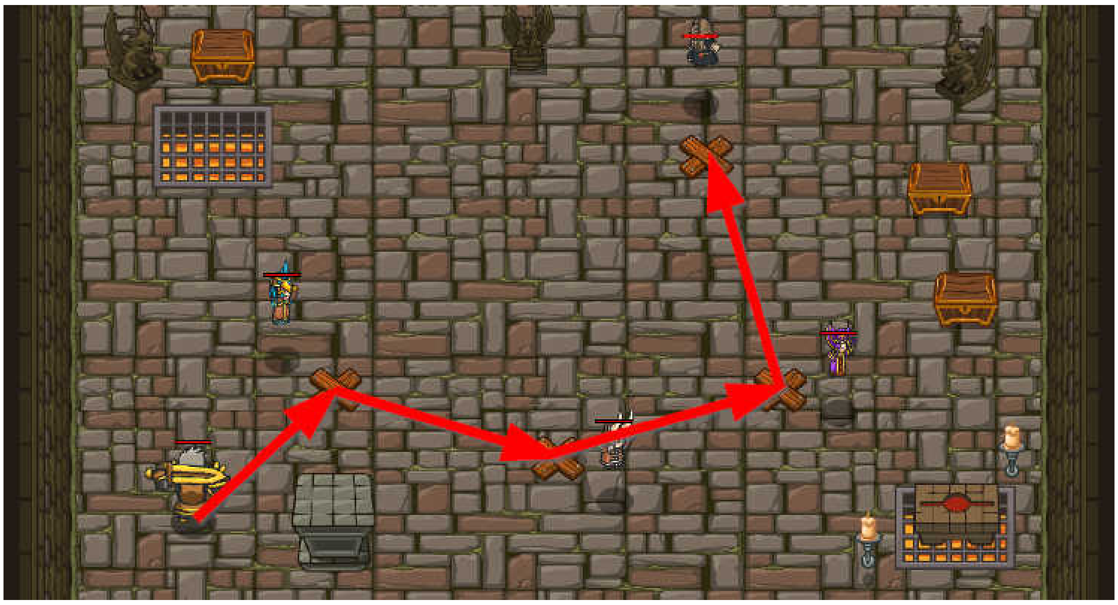
The Wizard's Door Solution

```
// Move to Laszlo and get his secret number.
hero.moveXY(30, 13);
var las = hero.findNearestFriend().getSecret();

// Add 7 to Laszlo's number to get Erzsebet's number.
// Move to Erzsebet and say her magic number.
var erz = las + 7;
hero.moveXY(17, 26);
hero.say(erz);

// Divide Erzsebet's number by 4 to get Simonyi's number.
// Go to Simonyi and tell him his number.
var sim = erz / 4;
hero.moveXY(30, 39);
hero.say(sim);

// Multiply Simonyi's number by Laszlo's to get Agata's number.
// Go to Agata and tell her her number.
var aga = sim * las;
hero.moveXY(43, 26);
hero.say(aga);
```

#5. The Wizard's Haunt

Level Overview and Solutions

Intro



You've gained entry to the wizards' hidden hideout! But you've heard rumors of a gateway to a secret chamber in a parallel dimension, containing untold treasure. Maybe you can talk your way into it.

Listen carefully to what each wizard says to figure out how to calculate each magic number. You'll need to take things one wizard at a time.

Look at the Hints to learn how to write out math formulas in code.

Default Code

```
// Move to Zsofia and get the secret number from her.
hero.moveXY(18, 20);
var zso = hero.findNearestFriend().getSecret();

// Divide Zsofia's number by 4 to get Mihaly's number.
// Move to Mihaly and say his magic number.
var mih = zso / 4;
hero.moveXY(30, 15);
hero.say(mih);

// Divide Mihaly's number by 5 to get Beata's number
// Move to Beata and say her magic number.

// Subtract Beata's number from Mihaly's to get Sandor's number.
// Move to Sandor and say his magic number.
```

Overview

In this level, you walk around to each wizard, following their instructions to calculate the magic numbers you need to speak. You'll have to edit and run your program multiple times to get all the instructions; take it one step at a time and you'll work your way through it.

You remember the syntax for add, subtrct, multiply, and modulo from "The Wizards Door:"

```
var a = 4 + 5;   // a = 9
var b = 7 - 3;   // b = 4
var c = 2 * 6;   // c = 12
var d = 8 % 3;   // d = 2
```

To divide, you just use the slash ('/') symbol:

```
var a = 6 / 2;   // a = 3
```

But what do you get when the numbers don't divide evenly?

```
var a = 6 / 5;   // a = 1.2
```

Don't forget to use variables to hold the wizards' magic numbers as you go through the level. You'll need to keep track of them to complete this level!

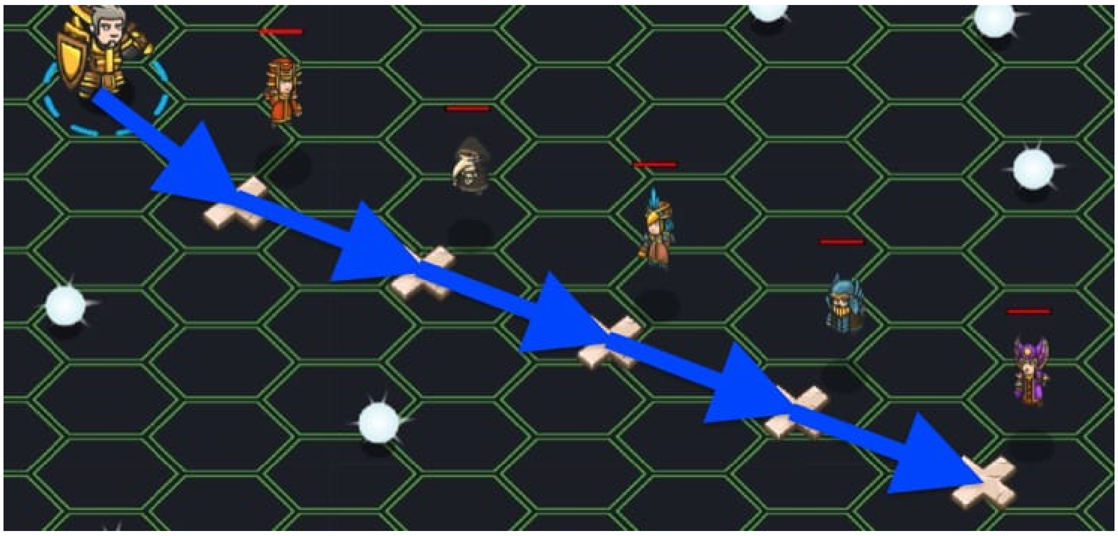
The Wizard's Haunt Solution

```
// Move to Zsofia and get the secret number from her.
hero.moveXY(18, 20);
var zso = hero.findNearestFriend().getSecret();

// Divide Zsofia's number by 4 to get Mihaly's number.
// Move to Mihaly and say his magic number.
var mih = zso / 4;
hero.moveXY(30, 15);
hero.say(mih);

// Divide Mihaly's number by 5 to get Beata's number
// Move to Beata and say her magic number.
var bea = mih / 5;
hero.moveXY(42, 20);
hero.say(bea);

// Subtract Beata's number from Mihaly's to get Sandor's number.
// Move to Sandor and say his magic number.
var san = mih - bea;
hero.moveXY(38, 37);
hero.say(san);
```

#6. The Wizard's Plane

Level Overview and Solutions

Intro



You have talked your way into a mysterious alternate plane of reality. Only the cleverest of wizards dwell here, but you're determined to find their secret treasures.

These crafty wizards weave complex spells, and will accept no mistakes from intruding adventuers. Use parentheses to make sure your formulae are in the correct order!

Listen carefully to what each wizard says to figure out how to calculate each magic number. You'll need to take things one wizard at a time.

Default Code

```
// Move to Eszter and get the secret number from her.
hero.moveXY(16, 32);
var esz = hero.findNearestFriend().getSecret();

// Multiply by 3 and subtract 2 to get Tamas's number.
// Remember to use parentheses to do calculations in the right order.
// Move to Tamas and say his magic number.
var tam = esz * 3 - 2;
hero.moveXY(24, 28);
hero.say(tam);

// Subtract 1 and multiply by 4 to get Zsofi's number.
// Move to Zsofi and say her magic number.

// Add Tamas's and Zsofi's numbers, then divide by 2 to get Istvan's number.
// Move to Istvan and say his magic number.

// Add Tamas's and Zsofi's numbers. Subtract Istvan's number from Zsofi's. Multiply the two results to
    get Csilla's number.
// Move to Csilla and say her magic number.
```

Overview

In this level, you walk to each wizard, following their instructions to calculate the magic numbers you need to speak. You'll have to edit and run your program multiple times to get all the instructions; take it one step at a time and you'll work your way through it.

You've learned how to write all kinds of arithmetic in code:

```
var a = 7 - 3;   // a = 4
var b = 2 * 6;   // b = 12
var c = 6 / 5;   // c = 1.2
```

In this level, you'll be combining these operations to do complex calculations. Make sure you use parentheses to ensure that things happen in the right order! Remember that multiplication and division come before addition and subtraction:

```
var a = 2 + 3 * 4;    # a = 14
var b = (2 + 3) * 4;  # b = 20
```

Don't forget to use variables to hold the wizards' magic numbers as you go through the level. You'll need to keep track of them to complete this level!

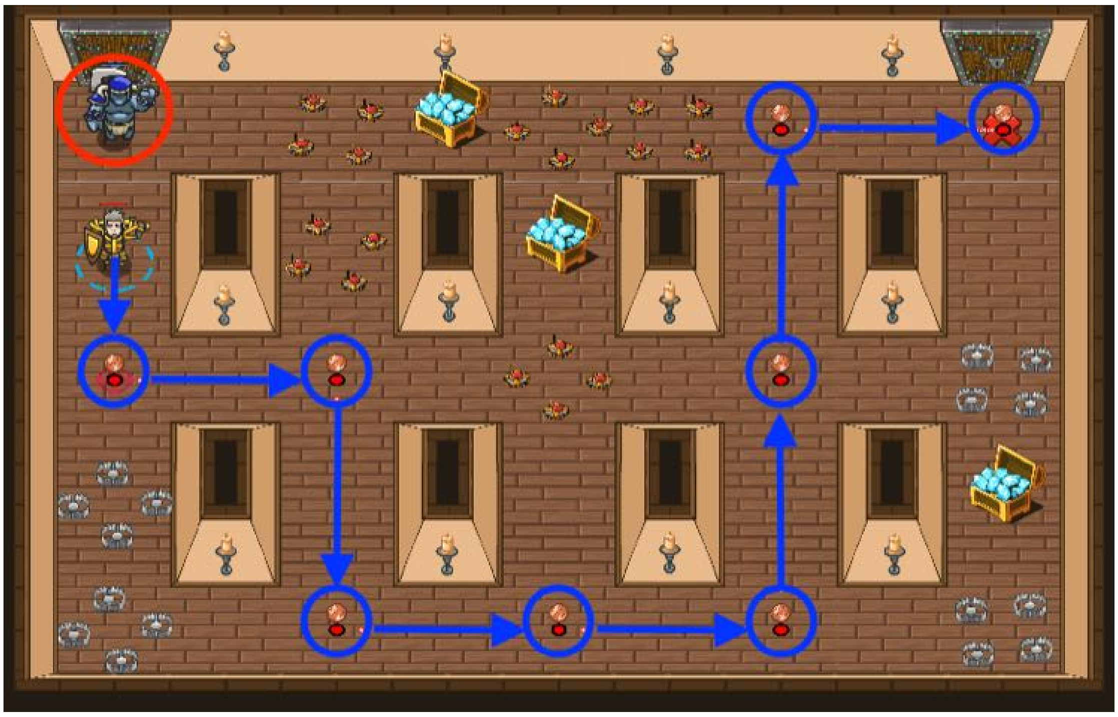
The Wizard's Plane Solution

```
// Move to Eszter and get the secret number from her.
hero.moveXY(16, 32);
var esz = hero.findNearestFriend().getSecret();

// Multiply by 3 and subtract 2 to get Tamas's number.
// Remember to use parentheses to do calculations in the right order.
// Move to Tamas and say his magic number.
var tam = esz * 3 - 2;
hero.moveXY(24, 28);
hero.say(tam);

// Subtract 1 and multiply by 4 to get Zsofi's number.
// Move to Zsofi and say her magic number.
var zso = (tam - 1) * 4;
hero.moveXY(32, 24);
hero.say(zso);

// Add Tamas's and Zsofi's numbers, then divide by 2 to get Istvan's number.
// Move to Istvan and say his magic number.
var ist = (tam + zso) / 2;
hero.moveXY(40, 20);
hero.say(ist);

// Add Tamas's and Zsofi's numbers. Subtract Istvan's number from Zsofi's. Multiply the two results to
    get Csilla's number.
// Move to Csilla and say her magic number.
var csi = (tam + zso) * (zso - ist);
hero.moveXY(48, 16);
hero.say(csi);
```

#7. Coincrumbs

Level Overview and Solutions

Intro



Can you escape from the storeroom without being caught by the ogre guard? Follow the trail of coins.

Just like checking an `enemy`'s `type`, you can check the position or `pos` of an `item`. You can even find the `x` and `y` coordinates of a `item`'s `pos`.

Mazes change when you click "Submit". Use items' positions instead hard-coding coordinates.

Default Code

```
// Follow the trail of coins to the red X at the exit.

while (true) {
    // This finds the nearest item.
    var item = hero.findNearestItem();
    if (item) {
        // This stores the item's pos, or position in a variable.
        var itemPosition = item.pos;
        // Put the X and Y coordinates of the item into variables.
        var itemX = itemPosition.x;
        var itemY = itemPosition.y;
        // Now, use moveXY to move to itemX and itemY:

    }
}
```

Overview

Your glasses have the `findNearestItem` method, which lets your hero find the nearest coins (potions, gems etc), but only when they are in your line of sight.

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

```
var coin = hero.findNearestItem();
if (coin) {
    var position = coin.pos;
    var coinX = position.x;
    var coinY = position.y;
    hero.moveXY(coinX, coinY);
}
```

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/or `buildXY`.

Coincrumbs Solution

```
// Follow the trail of coins to the red X at the exit.

while (true) {
    // This finds the nearest item.
    var item = hero.findNearestItem();
    if (item) {
        // This stores the item's pos, or position in a variable.
        var itemPosition = item.pos;
        // Put the X and Y coordinates of the item into variables.
        var itemX = itemPosition.x;
        var itemY = itemPosition.y;
        // Now, use moveXY to move to itemX and itemY:
        hero.moveXY(itemX, itemY);
    }
}
```

#8. White Rabbit

Level Overview and Solutions

Intro

The room is full of traps. Don't worry and follow the lightstone. The lightstone is an item and you can find it with the hero's `findNearestItem` method.

Try to collect the lightstone by moving to its position. It is fast but it will guide you to the exit.

Each item has the property `pos` which contains the items position. The object (`pos`) has properties `x` and `y` . Use them to find where you should move.

Default Code

```
// Follow the lightstone to navigate the traps.

while (true) {
    var item = hero.findNearestItem();
    if (item) {
        // Store the item's position in a new variable using item.pos:

        // Store the X and Y coordinates using pos.x and pos.y:

        // Move to the coordinates using moveXY() and the X and Y variables:

    }
}
```

Overview

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/or `buildXY`.

Also, you can reference `x` and `y` directly without using variables:

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

White Rabbit Solution

```
// Follow the lightstone to navigate the traps.

while (true) {
    var item = hero.findNearestItem();
    if (item) {
        // Store the item's position in a new variable using item.pos:
        var position = item.pos;
        // Store the X and Y coordinates using pos.x and pos.y:
        var x = position.x;
        var y = position.y;
        // Move to the coordinates using moveXY() and the X and Y variables:
        hero.moveXY(x, y);
    }
}
```

#9. Chameleons

Level Overview and Solutions

Intro

Ogres are hiding in this room. They can disguise as gems or coins. To find disguised ogres, you need to move really close to an item.

Collect all items and defend yourself from ogres when you see them. Use the nearest item's `pos` , `x` , and `y` to find where to move.

Default Code

```
// Ogres are disguised as coins or gems!

while (true) {
    var enemy = hero.findNearestEnemy();
    // If you see an enemy - attack it:

    var item = hero.findNearestItem();
    // If you see a coin or a gem - move to it's X and Y position:

}
```

Overview

Remember items have **objects** and they have **properties** like `pos` . In fact, `pos` is also an **object** and contains another 2 **properties** like `x` and `y` .

For example, if you wanted to find a friend's hair length and color:

```
var friend = hero.findNearestFriend();
if(friend) {
    var hair = friend.hair; // Access the friend's hair property
    var hairLength = hair.size; // Access the hair property's size property.
    var hairColor = hair.color; // Access the hair property's color property.
    hero.say("You have " + hairLength +  " "  + hairColor + " colored hair!");
}
```

Use this to move to the various coins's X and Y positions!

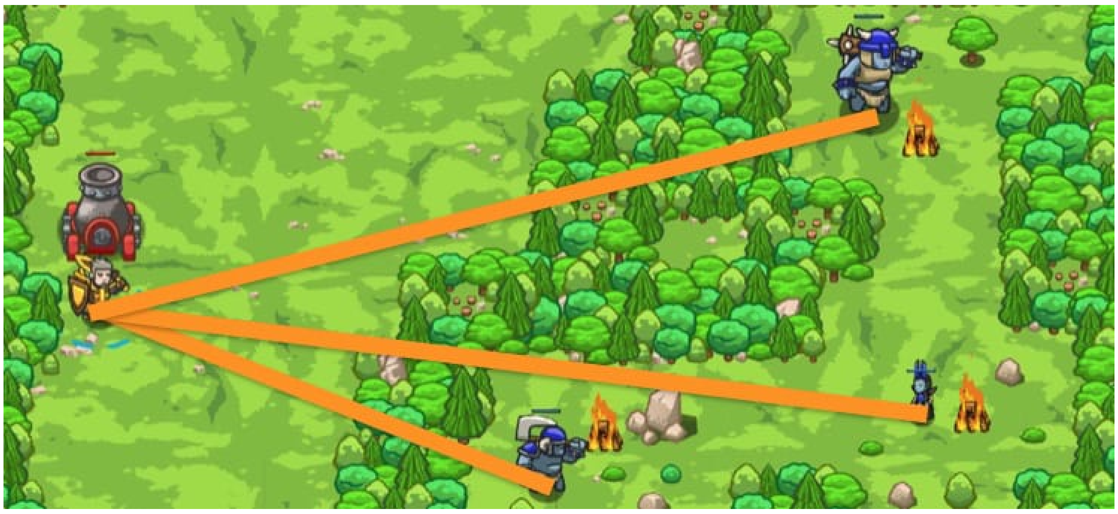
Chameleons Solution

```
// Ogres are disguised as coins or gems!

while (true) {
    var enemy = hero.findNearestEnemy();
    // If you see an enemy - attack it:
    if (enemy) {
        hero.attack(enemy);
    }
    var item = hero.findNearestItem();
    // If you see a coin or a gem - move to it's X and Y position:
    if (item) {
        var pos = item.pos;
        hero.moveXY(pos.x, pos.y);
    }
}
```

#10. Backwoods Bombardier

Level Overview and Solutions

Intro



Enemies (and heroes!) have a `pos` property that represents their position.

The `pos` property itself has two properties: `x` and `y` , which are floating-point (decimal) numbers:

```
var pos = enemy.pos;
var x = pos.x;
var y = pos.y;
```

Default Code

```
// The pos property is an object with x and y properties.
// pos.x is a number representing the horizontal position on the map
// pos.y is a number representing the vertical position on the map

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        var x = enemy.pos.x;
        // Get the enemy's y position!
        var y = 0; // ∆ Change this!

        // say the x and y position separated by a comma
        hero.say(x + ',' + y);
    } else {
        hero.say("Cease" + " Fire!");
    }
}
```

Overview

Enemies (and heroes!) have a `pos` property that represents their position.

The `pos` property itself has two properties: `x` and `y` , which are floating-point (decimal) numbers:

```
var pos = enemy.pos;
var x = pos.x;
var y = pos.y;
hero.say(x + "," + y)
```

The X represents the **horizontal** position of the enemy.

The Y represents the **vertical** position of the enemy.

Backwoods Bombardier Solution

```
// The pos property is an object with x and y properties.
// pos.x is a number representing the horizontal position on the map
// pos.y is a number representing the vertical position on the map

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        var x = enemy.pos.x;
        var y = enemy.pos.y;
        // say the x and y position separated by a comma
        hero.say(x + "," + y);
    } else {
        hero.say("Cease" + " Fire!");
    }
}
```

#11. Coinucopia

Level Overview and Solutions

Intro

You now have access to `flags` . Check the `Hints` for more information.

You don't need to change the sample code to win. Press `submit` and place some `flags` !

Default Code

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

Overview

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.

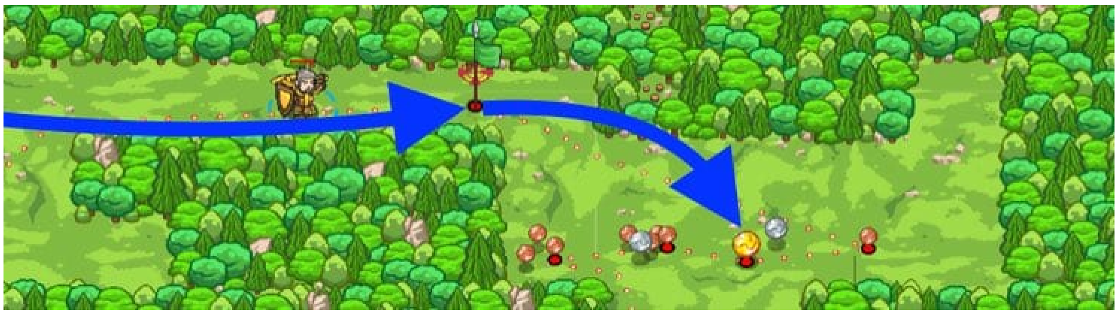
Coinucopia Solution

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

#12. Copper Meadows

Level Overview and Solutions

Intro



Just like checking an `enemy`'s `type`, you can check the position or `pos` of a `flag`. You can even find the `x` and `y` coordinates of a `flag`'s `pos`.

```
var flag = hero.findFlag();
hero.say(flag.pos);
hero.say(flag.pos.x);
```

Default Code

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

while(true) {
    var flag = hero.findFlag();
    if (flag) {
        // Pick up the flag.

    } else {
        // Automatically move to the nearest item you see.
        var item = hero.findNearestItem();
        if (item) {
            var position = item.pos;
            var x = position.x;
            var y = position.y;
            hero.moveXY(x, y);
        }
    }
}
```

Overview

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:

```
var item = hero.findNearestItem();
if (item) {
    var position = item.pos;
    var x = position.x;
    var y = position.y;
    hero.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.

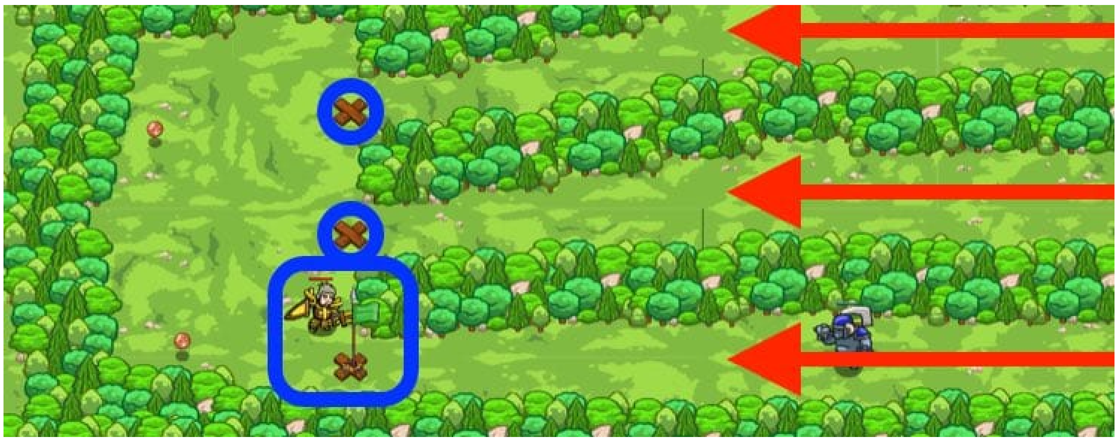
Copper Meadows Solution

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

while(true) {
    var flag = hero.findFlag();
    if(flag) {
        // Pick up the flag.
        hero.pickUpFlag(flag);
    } else {
        // Automatically move to the nearest item you see.
        var item = hero.findNearestItem();
        if(item) {
            var position = item.pos;
            var x = position.x;
            var y = position.y;
            hero.moveXY(x, y);
        }
    }
}
```

#13. Drop the Flag

Level Overview and Solutions

Intro



Before using `pickUpFlag()` , use `buildXY()` to build a `"fire-trap"` at the `flag` 's position.

A `coin` is like a `flag` , it has a `pos` . Use the `coin` 's `pos` to `moveXY()` to collect them.

Default Code

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

while(true) {
    var flag = hero.findFlag();
    if (flag) {
        // How do we get flagX and flagY from the flag's pos?
        // (Look below at how to get x and y from items.)


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

Overview

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

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

while(true) {
    var flag = hero.findFlag();
    if(flag) {
        // How do we get flagX and flagY 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;

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

#14. Mind the Trap

Level Overview and Solutions

Intro

Use `flags` and `distanceTo()` so you don't run over the mines!

Default Code

```
// 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!

while(true) {
    var flag = hero.findFlag();
    var enemy = hero.findNearestEnemy();

    if(flag) {
        // Pick up the flag.

        hero.say("I should pick up the flag.");
    } else if(enemy) {
        // Only attack if the enemy distance is < 10 meters

        hero.attack(enemy);
    }
}
```

Overview

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!

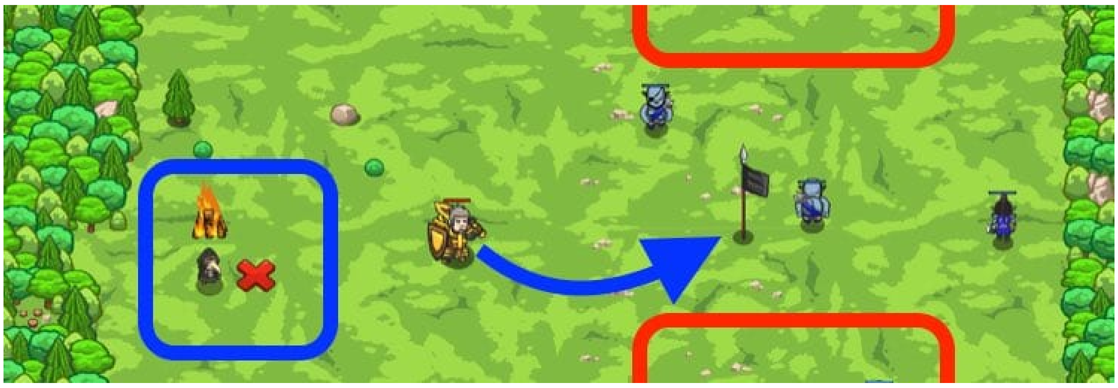
Mind the Trap Solution

```
// 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!

while(true) {
    var flag = hero.findFlag();
    var enemy = hero.findNearestEnemy();
    if(flag) {
        // Pick up the flag.
        hero.pickUpFlag(flag);
        hero.say("I should pick up the flag.");
    }
    else if (enemy) {
        // Only attack if the enemy distance is < 10 meters
        if(hero.distanceTo(enemy) < 10) {
            hero.attack(enemy);
        }
    }
}
```

#15. Signal Corpse

Level Overview and Solutions

Intro



Use different colored `flags` to tell your hero to run away or `cleave()` .

Default Code

```
// 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

while(true) {
    var green = hero.findFlag("green");
    var black = hero.findFlag("black");
    var nearest = hero.findNearestEnemy();

    if (green) {
        hero.pickUpFlag(green);
    } else if (black && hero.isReady("cleave")) {
        hero.pickUpFlag(black);
        // Cleave!

    } else if (nearest && hero.distanceTo(nearest) < 10) {
        // Attack!

    }
}
```

Overview

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

```
// 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

while(true) {
    var green = hero.findFlag("green");
    var black = hero.findFlag("black");
    var nearest = hero.findNearestEnemy();

    if(green) {
        hero.pickUpFlag(green);
    } else if (black && hero.isReady("cleave")) {
        hero.pickUpFlag(black);
        // Cleave!
        hero.cleave();
    } else if (nearest && hero.distanceTo(nearest) < 10) {
        // Attack!
        hero.attack(nearest);
    }
}
```

#16. Rich Forager

Level Overview and Solutions

Intro

Combine everything you know to venture through the groves! Remember `while-true loops`, `if/else`, `flags`, `cleave()`, `attack()`, `pos`, and `moveXY()`.

Default Code

```
// 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.

while(true) {
    var flag = hero.findFlag();
    var enemy = hero.findNearestEnemy();
    var item = hero.findNearestItem();

    if (flag) {
        // What happens when I find a flag?

    }
    else if (enemy) {
        // What happens when I find an enemy?

    }
    else if (item) {
        // What happens when I find an item?

    }
}
```

Overview

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 defeat 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.

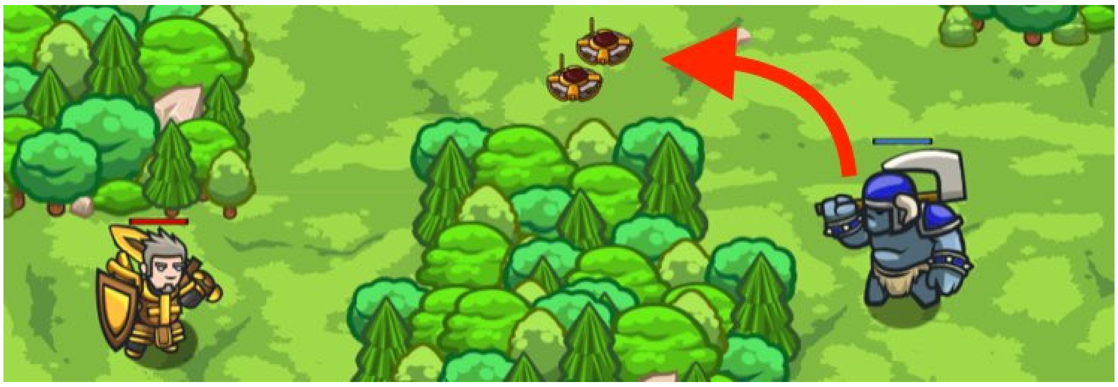
Rich Forager Solution

```
// 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.

while(true) {
    var flag = hero.findFlag();
    var enemy = hero.findNearestEnemy();
    var item = hero.findNearestItem();
    if(flag) {
        // What happens when I find a flag?
        hero.pickUpFlag(flag);
    } else if (enemy) {
        // What happens when I find an enemy?
        if(hero.isReady("cleave")) {
            hero.cleave(enemy);
        } else {
            hero.attack(enemy);
        }
    } else if (item) {
        // What happens when I find an item?
        hero.moveXY(item.pos.x, item.pos.y);
    }
}
```

#17. Thumb Biter

Level Overview and Solutions

Intro



Use the equality operator (==) to check if both sides match.

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

Default Code

```
// If-statement code only runs when the if’s condition is true.
// In a condition, == means "is equal to."
if (2 + 2 == 4) {
    hero.say("Hey!");
}
if (2 + 2 == 5) {
    hero.say("Yes, you!");
}

// Change the condition here to make your hero say "Come at me!"
if (3 + 3 == 7) {  // ∆ Make this true.
    hero.say("Come at me!");
}

if (2 == 20) {  // ∆ Make this true.
    // Add one more taunt to lure the ogre. Be creative!

}
```

Overview

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*"

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

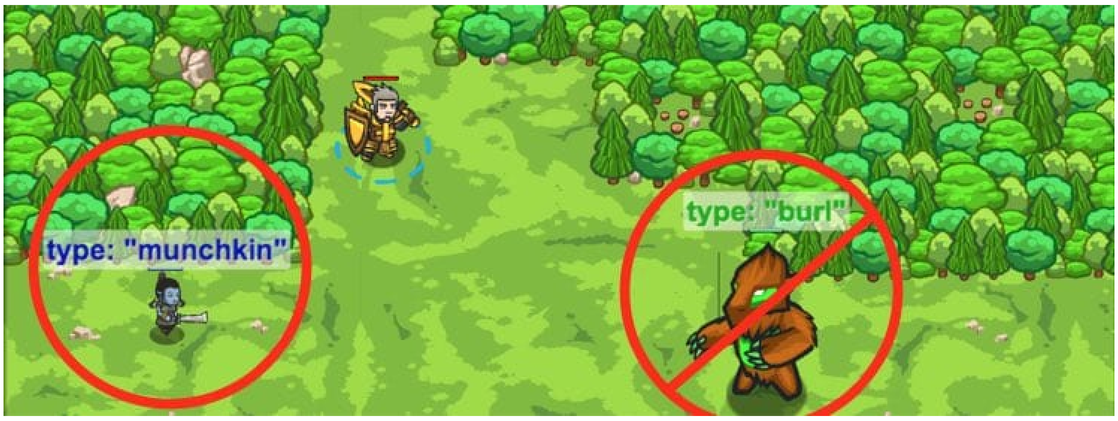
Thumb Biter Solution

```
// If-statement code only runs when the if’s condition is true.
// In a condition, == means "is equal to."
if(2 + 2 == 4) {
    hero.say("Hey!");
}
if(2 + 2 == 5) {
    hero.say("Yes, you!");
}

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

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

#18. Endangered Burl

Level Overview and Solutions

Intro



Learn more about your enemies by checking their `type` .

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

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

Default Code

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

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

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

    // Use "if" to attack a "thrower".


    // If it's an "ogre", use moveXY to run away to the village gate!


}
```

Overview

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:

```
var enemy = hero.findNearestEnemy();
if(enemy.type == "munchkin") {
    hero.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

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

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

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

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

#19. Burlbole Grove

Level Overview and Solutions

Intro

Functions can `return` a value!

When a function is called, it will be equal to whatever value it `returns` .

```
function plusTwo(x) {
    return x + 2;
}

var number = plusTwo(5);
// number is now 7
```

Default Code

```
// Don't attack the burls!
// Functions can return a value.
// When a function is called, it will be equal to the value the function returns.

function shouldAttack(target) {
    // return false if no target

    // return false if target.type == "burl"

    // Otherwise, return true
    return true;
}

while(true) {
    var enemy = hero.findNearestEnemy();
    // Here we use shouldAttack() to decide if we should attack!
    // heroShouldAttack will be assigned the same value that shouldAttack() returns!
    var heroShouldAttack = shouldAttack(enemy);
    if(heroShouldAttack) {
        hero.attack(enemy);
    }
}
```

Overview

Functions can `return` a value!

When a function is called, it will be equal to whatever value it `returns` .

```
function plusTwo(x) {
    return x + 2;
}

var number = plusTwo(5);
// number is now 7
```

In this level, your function `shouldAttack(target)` needs to return true if there is a valid target to attack, or false if not.

Then, your code can use `shouldAttack` to decide if your hero should attack!

```
function shouldAttack(target) {
    // return false if there is no target
    if(!target) {
        return false;
    }
    // Also return false if target.type == "burl"

    // Otherwise, return true
    return true;
}
```

NOTE: When a function executes a `return` statement, that function immediately ends!

```
function foo() {
    return "foo";
    hero.say("bar"); // This will never happen!
}
```

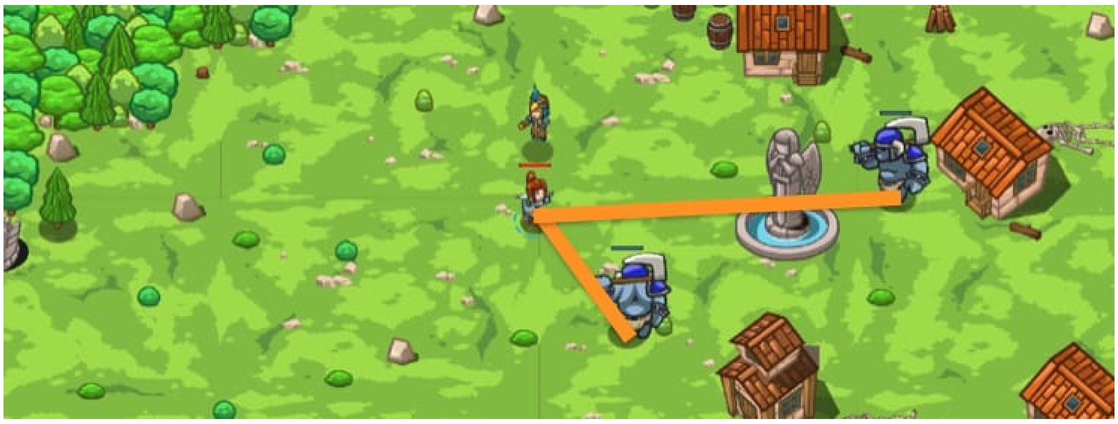
Burlbole Grove Solution

```
// Don't attack the burls!
// Functions can return a value.
// When a function is called, it will be equal to the value the function returns.

function shouldAttack(target) {
    // return false if no target
    if(!target) {
        return false;
    }
    // return false if target.type == "burl"
    if(target.type == "burl") {
        return false;
    }
    // Otherwise, return true
    return true;
}

while(true) {
    var enemy = hero.findNearestEnemy();
    // Here we use shouldAttack() to decide if we should attack!
    // heroShouldAttack will be assigned the same value that shouldAttack() returns!
    var heroShouldAttack = shouldAttack(enemy);
    if(heroShouldAttack) {
        hero.attack(enemy);
    }
}
```

#20. Blind Distance

Level Overview and Solutions

Intro



That village is too quiet. Looks like it's an ambush. The blind wizard is your only friend, but he is a really powerful mage. You will be the spotter for him. Watch for ogres and say the distance for any incoming. The wizard's powers are limited, use them **only when see an ogre**.

Use the predefined function that finds the nearest enemy and returns the distance to it (or 0 if no enemy). You can use function result in your code if you store it in a variable.

```
var enemy = hero.findNearestEnemy();
```

Default Code

```
// Tell the wizard the distance to the coming ogres.

// This function finds the nearest enemy and returns the distance to it.
function nearestEnemyDistance() {
    var enemy = hero.findNearestEnemy();
    // If there is no enemy, the function returns 0.
    var result = 0;
    if (enemy) {
        result = hero.distanceTo(enemy);
    }
    return result;
}

while (true) {
    // Call nearestEnemyDistance() and
    // save the result in the variable enemyDistance.
    var enemyDistance = nearestEnemyDistance();
    // If the enemyDistance is greater than 0:

        // Say the value of enemyDistance variable.

}
```

Overview

Functions can contain several (or many) instructions to make you can clearer and more readable. Also, functions allow avoiding repetition of code.

The function can return values and you can use them to get some data from them. You've met it before, when you used `hero.findNearestEnemy()`.

To return a value from a function, use the keyword `return` in the function. Place the value (or variable) which you want to return after that.

```
function someFunction() {
    ...
    return 3; // the function returns 3.
}
```

You can save function's result in a variable and use it further in your code:

```
var x = someFunction();
// Now x equals 3
hero.say(x);
```

Blind Distance Solution

```
// Tell the wizard the distance to the coming ogres.

// This function finds the nearest enemy and returns the distance to it.
// If there is no enemy, the function returns 0.
function nearestEnemyDistance() {
    var enemy = hero.findNearestEnemy();
    var result = 0;
    if (enemy) {
        result = hero.distanceTo(enemy);
    }
    return result;
}

while (true) {
    // Call nearestEnemyDistance() and
    // save the result in the variable enemyDistance.
    var enemyDistance = nearestEnemyDistance();
    // If the enemyDistance is greater than 0:
    if (enemyDistance > 0) {
        // Say the value of enemyDistance variable.
        hero.say(enemyDistance);
    }
}
```

#21. Hit and Freeze

Level Overview and Solutions

Intro

You're caught in a trap! Wait until the ogres are close, then attack, or you'll injure yourself!

Functions can return a value, including a `boolean` value (true or false).

Use this to decide if an ogre is `inAttackRange()`!

```
function inAttackRange(enemy) {
    var distance = hero.distanceTo(enemy);
    if(distance <= 3) {
        // return true because the enemy is in range
    } else {
        // return false because the enemy is out of range
    }
}
```

Save the result to a variable to use it later in the code:

```
var canAttack = inAttackRange(target);
```

Default Code

```
// You are trapped. Don't move, it'll be painful.

// This function checks if the enemy is in your attack range.
function inAttackRange(enemy) {
    var distance = hero.distanceTo(enemy);
    // Almost all swords have attack range of 3.
    if (distance <= 3) {
        return true;
    } else {
        return false;
    }
}

// Attack ogres only when they're within reach.
while (true) {
    // Find the nearest enemy and store it in a variable.

    // Call inAttackRange(enemy), with the enemy as the argument
    // and save the result in the variable canAttack.

    // If the result stored in canAttack is true, then attack!

}
```

Overview

You can have several `return` statements in a function, but only one will be used, because `return` causes the function to stop executing, and "returns" back to where the function was called.

```
function moreThanTen(n) {
    // if 'n' greater than 10, then the function will return true.
    if (n > 10) {
        return true;
    }
    // Otherwise 'return' inside 'else' will be called and the function will return false.
    else {
        return false;
    }
}
var isSmall = moreThanTen(5); // isSmall === true
```

Hit and Freeze Solution

```
// You are trapped. Don't move, it'll be painful.

// This function checks if the enemy is in your attack range.
function inAttackRange(enemy) {
    var distance = hero.distanceTo(enemy);
    // Almost all swords have attack range of 3.
    if (distance <= 3) {
        return true;
    } else {
        return false;
    }
}

// Attack ogres only when they're within reach.
while (true) {
    // Find the nearest enemy and store it in a variable.
    var nearestEnemy = hero.findNearestEnemy();
    // Call inAttackRange(enemy), with the enemy as the argument
    // and save the result in the variable canAttack.
    var canAttack = inAttackRange(nearestEnemy);
    // If the result stored in canAttack is true, then attack!
    if (canAttack) {
        hero.attack(nearestEnemy);
    }
}
```

#22. Coin Hunter

Level Overview and Solutions

Intro

The famous hunter Senick agreed to train you! Coins appear and disappear after a short time. Only move to coins that are closer than **20 meters**.

Write a function to decide if you should run for a coin:

```
// coin is passed in as a parameter
function isCoinClose(coin) {
    // Return true if the coin is close
    // Else return false
}
```

Default Code

```
// To make the training more interesting Senick poisoned you.
// While you aren't moving the poison is harmless.

// This function should check if a coin is closer than 20m.
function isCoinClose(coin) {
    // Find the distance to the coin.

    // If the distance is less than 20:

        // Return true

    // Else:

        // Return false

}

while (true) {
    var item = hero.findNearestItem();
    if (item) {
        // If isCoinClose(item) returns true:
        if (isCoinClose(item)) {
            hero.moveXY(item.pos.x, item.pos.y);
        }
    }
}
```

Overview

You need to write a function which receives a parameter `coin` (an item), finds the distance from the hero to that coin and decides if it close enough.

To get the distance to a coin use:

```
var distance = hero.distanceTo(coin);
```

To decide if it's close enough (distance less than 20 metres) use:

```
if (distance < 20) {
    ...
}
```

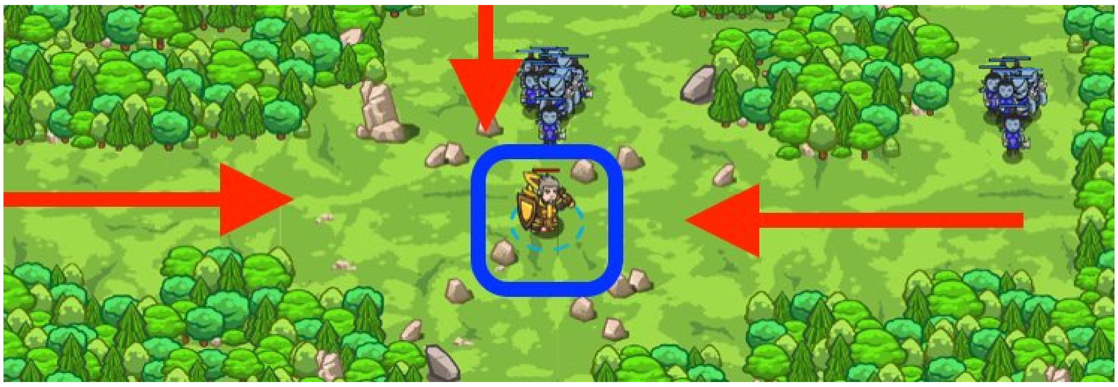
Coin Hunter Solution

```
// To make the training more interesting Senick poisoned you.
// While you aren't moving the poison is harmless.

// This function should check if a coin is closer than 20m.
function isCoinClose(coin) {
    // Find the distance to the coin.
    var distance = hero.distanceTo(coin);
    // If the distance is less than 20:
    if (distance < 20) {
        // Return true
        return true;
    }
    // Else:
    else {
        // Return false
        return false;
    }
}

while (true) {
    var item = hero.findNearestItem();
    if (item) {
        // If isCoinClose(item) returns true:
        if (isCoinClose(item)) {
            hero.moveXY(item.pos.x, item.pos.y);
        }
    }
}
```

#23. Agrippa Returned

Level Overview and Solutions

Intro



Just as in the previous "Agrippa Defense" level, you'll be replaying the same scenario, but with even more advanced functions.

You can use a function to calculate a value, then `return` it so you can use it in your code.

```
hero.triple = function(a) {
    var b = a + a + a;
    return b;
};

hero.say("3 × 3 = " + triple(3));
hero.say("4 × 3 = " + triple(4));
```

Default Code

```
function enemyInRange(enemy) {
    // Return true if the enemy is less than 5 units away.

    return false;
}

function cleaveOrAttack(enemy) {
    if (hero.isReady("cleave")) {
        hero.cleave(enemy);
    }
    else {
        hero.attack(enemy);
    }
}

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        // Check the distance of the enemy by calling enemyInRange.
        if (enemyInRange(enemy)) {
            cleaveOrAttack(enemy);
        }
    }
}
```

Overview

In the previous "Agrippa Defense" level, we streamlined our code by using functions to extract logic. In this level, we use a function to calculate a value and pass it back to the calling code.

In this level, we do a distance check so that we can only fight ogres that are close to us:

```
while (true) {
    var enemy = hero.findNearestEnemy();
    if (enemy) {
        var  = hero.distanceTo(enemy);
        if (distance < 5) {
            hero.cleaveOrAttack(enemy);
        }
    }
}
```

We can make this code cleaner by putting the distance check in its own function. The function `return`s a "true" or "false" value so we can decide whether or not to attack:

```
while (true) {
    var enemy = hero.findNearestEnemy();
    if (enemy) {
        if (hero.enemyInRange(enemy)) {
            hero.cleaveOrAttack(enemy);
        }
    }
}
```

When you have a function calculate a value, use the `return` keyword to end the function and send a value back to the caller:

```
hero.triple = function(a) {
    var b = a + a + a;
    return b;
};

hero.say("3 × 3 = " + triple(3));
hero.say("4 × 3 = " + triple(4));
```

Agrippa Returned Solution

```
hero.enemyInRange = function(enemy) {
    // Return true if the enemy is less than 5 units away.
    var distance = hero.distanceTo(enemy);
    if (distance < 5) {
        return true;
    } else {
        return false;
    }
};

hero.cleaveOrAttack = function(enemy) {
    if (hero.isReady("cleave")) {
        hero.cleave(enemy);
    } else {
        hero.attack(enemy);
    }
};

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        // Check the distance of the enemy by calling enemyInRange.
        if (hero.enemyInRange(enemy)) {
            hero.cleaveOrAttack(enemy);
        }
    }
}
```

#24. Metal Detector

Level Overview and Solutions

Intro

That ogre camp is well guarded, but we can reduce the ogre numbers. We hid the artillery in the forest and prepared gold coins. Our plan is simple: a coin appears, an ogre runs for the coin, we shoot at the coin, done.

You task is to work as a rangefinder. Wait for a coin and then `say` the `distanceTo()` the coin.

Default Code

```
// The artillery uses coins as a target.
// You'll be the rangefinder for the artillery.

// Write the function.
function coinDistance() {
    // Find the nearest coin,

    // If there is a coin, return the distance to it.

    // Else, return 0 (zero).

}

while (true) {
    var distance = coinDistance();
    if (distance > 0) {
        // Say the distance.

    }
}
```

Overview

You've learned about functions with `return` in the previous levels. Now it's the time to prove your knowledge! Complete the function and don't forget `return` in it.

To get the distance to an item use glasses method `hero.distanceTo(item)`. You function should return 0 (zero) if there isn't a coin.

Metal Detector Solution

```
// The artillery uses coins as a target.
// You'll be the rangefinder for the artillery.

// Write the function.
function coinDistance() {
    // Find the nearest coin,
    var coin = hero.findNearestItem();
    // If there is a coin, return the distance to it.
    if(coin) {
        var distance = hero.distanceTo(coin);
        return distance;
    } else {
    // Else, return 0 (zero).
        return 0;
    }
}

while (true) {
    var distance = coinDistance();
    if (distance > 0) {
        // Say the distance.
        hero.say(distance);
    }
}
```

#25. Passing Through

Level Overview and Solutions

Intro

You found a village of peaceful ogres. If you insult them, they will join the hostile ogres!

They will be insulted if you take their food, or if you don't take the gems they offer you.

The sample code shows you how to compare things with `!=`.

```
if(item.type != "gem) {
    // The item is not a "gem".
}
```

Default Code

```
// Don't insult this tribe of peaceful ogres.

while(true) {
    var item = hero.findNearestItem();
    if(item) {
        // If item.type IS NOT EQUAL TO "gem"
        if(item.type != "gem") {
            // Then follow your pet wolf.
            hero.moveXY(pet.pos.x, pet.pos.y);
        }
        // Else:

            // Move to the gem's position.

    }
}
```

Overview

This level shows you how to use `!=`.

You know that `==` means **is equal to**.

`!=` is similar, but it means **is NOT equal to**.

```
In Lua, `not equal to` is written `~=`
```

In this case, we've given you the code with `!=`, and what you have to do is write the `else` part of the code, using `moveXY` to move the hero to `item`s `pos.x` and `pos.y` position.

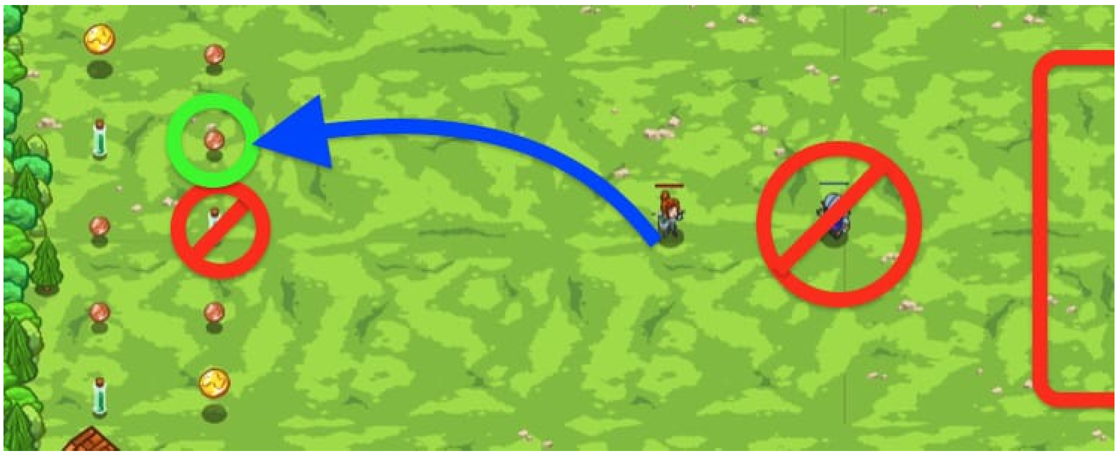
Passing Through Solution

```
// Don't insult this tribe of peaceful ogres.

while(true) {
    var item = hero.findNearestItem();
    if(item) {
        // If item.type IS NOT EQUAL TO "gem"
        if(item.type != "gem") {
            // Then follow your pet wolf.
            hero.moveXY(pet.pos.x, pet.pos.y);
        }
        // Else:
        else {
            // Move to the gem's position.
            hero.moveXY(item.pos.x, item.pos.y);
        }
    }
}
```

#26. Useful Competitors

Level Overview and Solutions

Intro



Check an item's `type` to make sure the hero doesn't pick up `"poison"`.

Default Code

```
// The coin field has been seeded with vials of deadly poison.
// Ogres are attacking, while their peons are trying to steal your coins!
// Attack the enemy only if the type is NOT equal to "peon".

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        if(enemy.type != "peon") {
            hero.attack(enemy);
        }
    }
    var item = hero.findNearestItem();
    if(item) {
        // Gather the item only if the type is NOT equal to "poison".

    }
}
```

Overview

In this level you'll need to use the `not` operator to filter your enemies and items!

`not` takes the logical inverse of a value and returns it:

```
// Not in JavaScript is written as: "!".
hero.say(!false); // The hero says 'true'
hero.say(!true); // The hero says 'false'
```

To use in a conditional statement:

```
if(!hero.isReady('cleave')) {
    // Do something while cleave is on cooldown!
} else {
    // Cleave is ready.
}
```

Useful Competitors Solution

```
// The coin field has been seeded with vials of deadly poison.
// Ogres are attacking, while their peons are trying to steal your coins!
// Attack the enemy only if the type is NOT equal to "peon".

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        if(enemy.type != "peon") {
            hero.attack(enemy);
        }
    }
    var item = hero.findNearestItem();
    if(item) {
        // Gather the item only if the type is NOT equal to "poison".
        if(item.type != "poison") {
            hero.moveXY(item.pos.x, item.pos.y);
        }
    }
}
```

#27. Wonderglade

Level Overview and Solutions

Intro

Wonderglade can give you any items that you need. You arrived here to collect some ingredients and coins. But you aren't the only one here. That burl likes shiny gems! The burl doesn't like if someone takes its gems.

Collect all items **except** gems (`item.type` is `"gem"`) The comparison operator `not equal to` (`!=`) can be useful for that.

Default Code

```
// You need to collect several items.
// But, the burl wants the gems!
// Pick up all appearing items EXCEPT gems.

while (true) {
    var item = hero.findNearestItem();
    if (item) {
        // If item.type isn't equal to "gem":

            // Move to the item's position.

    }
}
```

Overview

The operator `not equal to` is written as `!=`. It can be very useful when you have many 'positive' conditions and only one 'negative'. In this case instead of enumerating all conditions when you can do something, you can use `!=` once and describe the reverse case. Also, it can be used for cases when you don't know all 'positive' conditions, but you definitely know what you shouldn't do.

So instead to check all types those you need to pick up:

```
if (item.type == 'coin') {
    // Do something.

}
if (item.type == 'potion') {
    // Do something.

}
```

You can just to check for the non-equality for the certain restricted type:

```
if (item.type != 'gem') {
    // Do something

}
```

Wonderglade Solution

```
// You need to collect several items.
// But, the burl wants the gems!
// Pick up all appearing items EXCEPT gems.

while (true) {
    // Find the nearest item.
    var item = hero.findNearestItem();
    // If there is an item:
    if (item) {
        // If item.type isn't equal to "gem":
        if (item.type != "gem") {
            // Move to the item's position.
            hero.moveXY(item.pos.x, item.pos.y);
        }
    }
}
```

#28. Cursed Wonderglade

Level Overview and Solutions

Intro

We returned to Wonderglade, but it's changed. Ogres cursed the glade!

The burl is still here so don't touch gems. Collect all items **except** gems (`item.type` is `"gem"`) Also defeat all enemies **except** the burl (`enemy.type` is `"burl"`)

Default Code

```
// Wonderglade has changed since our last visit.
// Ogres cursed it and we should defeat them.
// The burl still is collecting gems, so don't touch them.
// Also don't attack the burl.

while (true) {
    // Find the nearest item.
    // Collect it (if it exists) only if its type isn't "gem".

    // Find the nearest enemy.
    // Attack it if it exists and its type isn't "burl".

}
```

Overview

You know how to use the operator `not equal to` (`!=`) from the previous levels. Use that knowledge to complete this level.

You need to write two blocks inside `while-loop` . The first one for items: if there is an item and its type `not equal to "gem"` , then collect it. The second one for enemies: if there is an enemy and its type `not equal to "burl"` , then attack it.

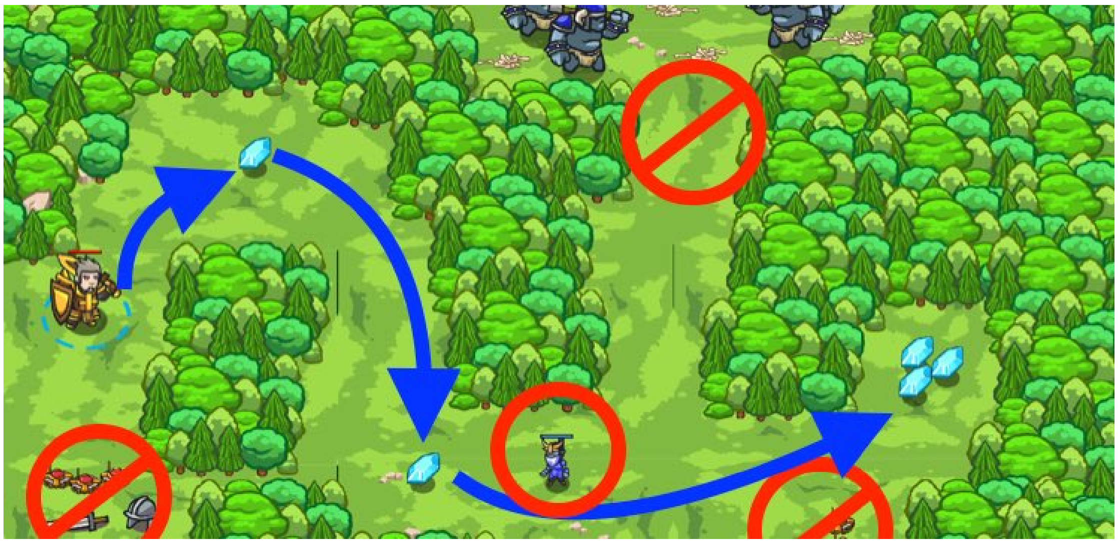
Cursed Wonderglade Solution

```
// Wonderglade has changed since our last visit.
// Ogres cursed it and we should defeat them.
// The burl still is collecting gems, so don't touch them.
// Also don't attack the burl.

while (true) {
    // Find the nearest item.
    // Collect it (if it exists) only if its type isn't "gem".
    var item = hero.findNearestItem();
    if (item) {
        if (item.type != 'gem'){
            hero.moveXY(item.pos.x, item.pos.y);
        }
    }
    // Find the nearest enemy.
    // Attack it if it exists and its type isn't "burl".
    var enemy = hero.findNearestEnemy();
    if (enemy) {
        if (enemy.type != 'burl') {
            hero.attack(enemy)
        }
    }
}
```

#29. Gems or Death

Level Overview and Solutions

Intro



Make the `if-statement` `true` for the commands you want to execute, and `false` for the commands you don't want to execute.

Remember the `==` operator means "is equal to?".

`1 + 2 == 3` is `true`.

`3 + 3 == 5` is `false`.

Default Code

```
// If-statement code only runs when the if’s condition is true.
// Fix all the if-statements to beat the level.

// == means "is equal to".
if (1 + 1 + 1 == 3) {  // Δ Make this false.
    hero.moveXY(5, 15);  // Move to the first mines.
}
if (2 + 2 == 5) {  // Δ Make this true.
    hero.moveXY(15, 40);  // Move to the first gem.
}
// != means "is not equal to".
if (2 + 2 != 4) {  // Δ Make this true.
    hero.moveXY(25, 15);  // Move to the second gem.
}
// < means "is less than".
if (2 + 2 < 3) {  // Δ Make this true.
    var enemy = hero.findNearestEnemy();
    hero.attack(enemy);
}
if (2 < 4) {  // Δ Make this false.
    hero.moveXY(40, 55);
}
if (true) {  // Δ Make this false.
    hero.moveXY(50, 10);
}
if (false) {  // Δ Make this true.
    hero.moveXY(55, 25);
}
```

Overview

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`:

```
if(1 + 1 + 1 == 3) {
    hero.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

```
// If-statement code only runs 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) { // Δ Make this false.
    hero.moveXY(5, 15); // Move to the first mines.
}

if(2 + 2 == 4) { // Δ Make this true.
    hero.moveXY(15, 40); // Move to the first gem.
}
// != means "is not equal to".
if(2+2 != 5) {
    hero.moveXY(25, 15); // Move to the second gem.
}
// < means "is less than".
if (2 + 2 < 5) { // Δ Make this true.
    var enemy = hero.findNearestEnemy();
    hero.attack(enemy);
}

if(2 < 1) { // Δ Make this false.
    hero.moveXY(40, 55);
}

if(false) { // Δ Make this false.
    hero.moveXY(50, 10);
}

if(true) { // Δ Make this true.
    hero.moveXY(55, 25);
}
```

#30. Burls Beets Booleans

Level Overview and Solutions

Intro

Answer `true` or `false` to the Burl's questions!

Be sure to read the **comments** above each line!

Default Code

```
// A Boolean is a value that is either true or false
// The == symbol means "is equal to?"
// So, writing A == B is asking the question: "A is equal to B?"
// The answer is a boolean value!
// Click the "Hints" button if this is confusing!

// Question: 2 == 3
// Say the correct answer:
hero.say(false);

// Question: 3 == 3
// Answer true or false for question 2:
hero.say("I don't know!");

// Question: "Three" == 3
// Answer true or false for question 3:
hero.say("I don't know!");

// Question: "Three" == "Three"
// Answer true or false for question 4:
hero.say("I don't know!");

// Question: 1 + 2 == 3
// Answer true or false for question 5:
hero.say("I don't know!");
```

Overview

What is a Boolean?

Boolean describes a type of data, just like String or Number describe different types of data.

A string is text, usually written inside of double quotes, like `"This is a string."`

A boolean is a value that is either **TRUE** or **FALSE**. It is written slightly differently depending on what programming language you use.

In Python a boolean is either `True` or `False` (with capitalized first letters).

In JavaScript and CoffeeScript, a boolean is either `true` or `false` (no capital letters!).

Why do we use booleans? Asking questions.

When coding, it's useful to determine if a question is true or false!

One type of question often asked is **EQUALITY**: "Is A equal to B?"

To ask this question in code, we use the **equality operator** which is usually written as `==`.

Think of `==` as meaning *"is equal to?"*. So To ask: "A is equal to? B" you write `A == B`.

Important Notes

Remember when we assigned values to a variable using code like: `enemy = hero.findNearestEnemy()` ?

Notice that the **assignment operator** is a single `=`.

The **equality operator** uses two: `==`.

Many tears have been shed by programmers trying to find a bug caused by using one `=` instead of two!

Also, note that the **string** `"true"` is not the same thing as the **boolean** `true`, though in this level, the Burl will accept your answer if you say either one. Burls are nice that way.

Burls Beets Booleans Solution

```
// A Boolean is a value that is either true or false
// The == symbol means "is equal to?"
// So, writing A == B is asking the question: "A is equal to B?"
// The answer is a boolean value!
// Click the "Hints" button if this is confusing!

// Question: 2 == 3
// Say the correct answer:
hero.say(false);

// Question: 3 == 3
// Answer true or false for question 2:
hero.say(true);

// Question: "Three" == 3
// Answer true or false for question 3:
hero.say(false);

// Question: "Three" == "Three"
// Answer true or false for question 4:
hero.say(true);

// Question: 1 + 2 == 3
// Answer true or false for question 5:
hero.say(true);
```

#31. Salted Earth

Level Overview and Solutions

Intro

This level introduces the concept of the `boolean or`.

Placing an `or` between two boolean values will return a single boolean value, much like `+` takes 2 numbers and spits out a another number.

`or` returns `true` if either the value before or after is `true`, or `false` if both are `false`.

```
// Write boolean or using '||'
hero.say(false || false); // Hero says 'false'
hero.say(false || true); // Hero says 'true'
hero.say(true || false); // Hero says 'true'
hero.say(true || true); // Hero says 'true'
```

Default Code

```
// Ogres are attacking a nearby settlement!
// Be careful, though, for the ogres have sown the ground with poison.
// Gather coins and defeat the ogres, but avoid the burls and poison!

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy.type == "munchkin" || enemy.type == "thrower") {
        hero.attack(enemy);
    }
    var item = hero.findNearestItem();
    // Check the item type to make sure the hero doesn't pick up poison!
    // Look for types: 'gem' and 'coin'

}
```

Overview

This level introduces the concept of the `boolean or`. Placing an `or` between two boolean values will return a single boolean value, much like `+` takes 2 numbers and spits out a another number (in this case, the sum).

Remember that booleans are a single bit of data, `true` or `false`. `or` returns `true` if either the value before or after is `true`, or `false` if both are `false`.

```
// Write boolean or using '||'
hero.say(false || false); // Hero says 'false'
hero.say(false || true); // Hero says 'true'
hero.say(true || false); // Hero says 'true'
hero.say(true || true); // Hero says 'true'
```

Which is useful if you know the exact boolean, but, programming lets you do so much more!

Recall that `<` , `>` , `<=` , `>=` , `==` return boolean values, so to make this more useful:

```
var enemy = hero.findNearestEnemy();
// It helps to read it outloud:
if(hero.distanceTo(enemy) < 10 || enemy.type == "thrower") {
    // If distanceTo enemy is less than 10, OR, enemy type is thrower
    hero.attack(enemy)
}
```

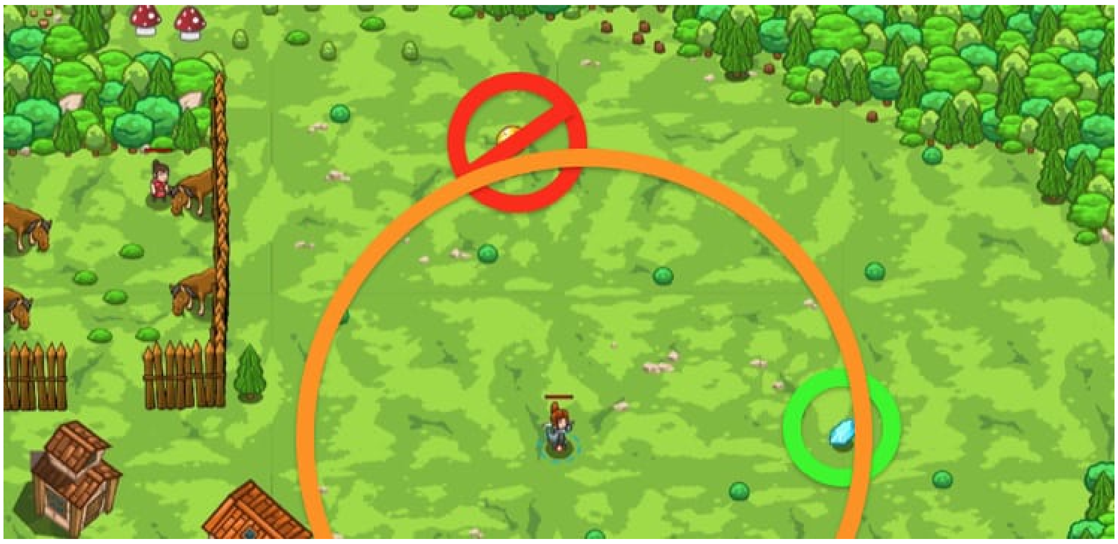
Salted Earth Solution

```
// Ogres are attacking a nearby settlement!
// Be careful, though, for the ogres have sown the ground with poison.
// Gather coins and defeat the ogres, but avoid the burls and poison!

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy.type == "munchkin" || enemy.type == "thrower") {
        hero.attack(enemy);
    }
    var item = hero.findNearestItem();
    // Check the item type to make sure the hero doesn't pick up poison!
    // Look for types: 'gem' and 'coin'
    if(item.type == "coin" || item.type == "gem") {
        hero.moveXY(item.pos.x, item.pos.y);
    }
}
```

#32. Star Shower

Level Overview and Solutions

Intro



A star shower is raining gems and coins down on you! But star metal isn't long-lived and coins disappear quickly. Gems don't disappear.

Use an **OR** statement to pick up close coins, OR gems:

```
if(item.type == "gem" || distance < 20) {
    // Get the item!
}
```

P.S. **Don't eat mushrooms!**

Default Code

```
// Pick up coins only if they are closer than 20m.
// Pick up all gems.

while (true) {
    var item = hero.findNearestItem();
    var distance = hero.distanceTo(item);
    // If the item's type is "gem"
    // OR the distance to the item less than 20 meters:

        // Move to item's position.

}
```

Overview

The logical **OR** operator can make your code readable and help to avoid repetition. For example instead several `if` statements:

```
if (condition1) {
    // Do something

}
if (condition2) {
    // Do the same again

}
```

you can put them in one:

```
if (condition1 || condition2) {
    // Do something

}
```

Avoiding to repeat the same code is a good practice because it makes your code readable. Also if you want to change some code and logic you can do it one place.

Star Shower Solution

```
// Pick up coins only if they are closer than 20m.
// Pick up all gems.

while (true) {
    var item = hero.findNearestItem();
    var distance = hero.distanceTo(item);
    // If the item's type is "gem"
    // OR the distance to the item less than 20 meters:
    if (item.type == "gem" || distance < 20) {
        // Move to item's position.
        hero.moveXY(item.pos.x, item.pos.y);
    }
}
```

#33. Forest Shadow

Level Overview and Solutions

Intro

This grove is a nice place for an ambush. The big ogres can't see you through all the trees, so hunt for ogre `"thrower"` s or `"munchkin"` s only.

Also, it's a good opportunity to collect shiny items - `"gem"` s or `"coin"` s.

Don't drink or eat anything!

Default Code

```
// Big ogres can't see you in the forest.
// Attack only the small ogres in the forest.
// Collect coins and gems only.
// Don't leave the forest and don't eat/drink anything.

while (true) {
    // Find the nearest enemy.

    // Attack it only if its type is "thrower" or "munchkin".

    // Find the nearest item.

    // Collect it only if its type is "gem" or "coin".

}
```

Overview

Remember you can use an **OR** statement to find different kinds of enemies or items:

```
if(enemy.type == "munchkin" || enemy.type == "thrower") {
    hero.attack(enemy)
}
```

Forest Shadow Solution

```
// Big ogres can't see you in the forest.
// Attack only the small ogres in the forest.
// Collect coins and gems only.
// Don't leave the forest and don't eat/drink anything.

while (true) {
    // Find the nearest enemy.
    // Attack it only if its type is "thrower" or "munchkin".
    var enemy = hero.findNearestEnemy();
    if (enemy.type == "thrower" || enemy.type == "munchkin") {
        hero.attack(enemy);
    }
    // Find the nearest item.
    // Collect it only if its type is "gem" or "coin".
    var item = hero.findNearestItem();
    if (item.type == "gem" || item.type == "coin") {
        hero.moveXY(item.pos.x, item.pos.y);
    }
}
```

#34. Spring Thunder

Level Overview and Solutions

Intro

Treasure collecting is a dangerous work when it's a thunderstorm outside. Some gems and coins attract lightning - you'll want to avoid those!

Use an AND operator to determine if an item is safe to pick up.

```
var item = hero.findNearestItem();
if(item.type == "coin" && item.value == 2) {
    hero.moveXY(item.pos.x, item.pos.y);
}
```

`A AND B` is true only if both A and B are true.

Default Code

```
// Certain coins and gems attract lightning.
// The hero should only grab silver coins and blue gems.

while (true) {
    var item = hero.findNearestItem();
    // A silver coin has a value of 2.
    // Collect if item.type is equal to "coin"
    // AND item.value is equal to 2.
    if (item.type == "coin" && item.value == 2) {
        hero.moveXY(item.pos.x, item.pos.y);
    }
    // A blue gem has a value of 10.
    // Collect if item.type is equal to "gem"
    // AND item.value is equal to 10.

}
```

Overview

This level introduces the concept of the `boolean and`.

Placing an `and` between two boolean values will return a single boolean value, much like `*` takes 2 numbers and spits out an another number (in this case, the multiplication).

Remember that booleans are a single bit of data, `true` or `false`.

`and` returns `true` if both the value before and after is `true`, or `false` if one (or both) of them is `false`.

```
// Write boolean 'and' using '&&'
hero.say(false && false); // Hero says 'false'
hero.say(false && true); // Hero says 'false'
hero.say(true && false); // Hero says 'false'
hero.say(true && true); // Hero says 'true'
```

Recall that `<`, `>`, `<=`, `>=`, `==` return boolean values, so to make this more useful:

```
var item = hero.findNearestItem();
// It helps to read it out loud:
if(hero.distanceTo(item) < 15 && item.type == "potion") {
    // If distanceTo the item is less than 15, AND, item's type is a potion
    hero.moveXY(item.pos.x, item.pos.y);
}
```

Spring Thunder Solution

```
// Certain coins and gems attract lightning.
// The hero should only grab silver coins and blue gems.

while (true) {
    var item = hero.findNearestItem();
    // A silver coin has a value of 2.
    // Collect if item.type is equal to "coin"
    // AND item.value is equal to 2.
    if (item.type == "coin" && item.value == 2) {
        hero.moveXY(item.pos.x, item.pos.y);
    }
    // A blue gem has a value of 10.
    // Collect if item.type is equal to "gem"
    // AND item.value is equal to 10.
    if (item.type == "gem" && item.value == 10) {
        hero.moveXY(item.pos.x, item.pos.y);
    }
}
```

#35. Teleport Lasso

Level Overview and Solutions

Intro

The ogre army is strong, so we are going to teleport them here, one by one. The teleportation isn't stable, so ogres appear for a short time. If you attack an ogre, you will make it stable.

Attack only weak ogres - `"munchkin"` s, and only if they're closer than 20 meters.

Use an AND operator to make sure both of these conditions are true!

Default Code

```
// Our wizards teleport ogres from their camp here.
// They appear for a short period and they are stunned.
// Attack only weak and near ogres.

while (true) {
    var enemy = hero.findNearestEnemy();
    var distance = hero.distanceTo(enemy);
    // If enemy.type is "munchkin"
    // AND the distance to it is less than 20m

        // Then attack it.

}
```

Overview

The operator `and` (`&&`) takes two boolean operands and return `true` only if both operands are `true` .

```
// If item's type is "coin" AND its value is 2
if (item.type == "coin" && item.value == 2) {
    // Do something

}
```

In this level, attack an ogre if its `type` is `"munchkin"` AND the distance to it is **less than 20 metres**.

Teleport Lasso Solution

```
// Our wizards teleport ogres from their camp here.
// They appear for a short period and they are stunned.
// Attack only weak and near ogres.

while (true) {
    var enemy = hero.findNearestEnemy();
    var distance = hero.distanceTo(enemy);
    // If enemy.type is "munchkin"
    // AND the distance to it is less than 20m
    if (enemy.type == "munchkin" && distance < 20) {
        // Then attack it.
        hero.attack(enemy);
    }
}
```

#36. Brawler Hunt

Level Overview and Solutions

Intro

Our archers are ready to defend against most ogres, but they can't defeat the huge ogre `"brawler"` s. Against those big guys, we have artillery.

Ammunition is low, but it should be enough to stop all brawlers. Don't waste cannon shells against other ogres.

If an ogre is type `"brawler"` AND the distance to it is less than 50 meters, command the artillery to fire by saying `"Fire!"`.

Default Code

```
// Don't worry about small and medium-sized ogres.
// Your targets are type "brawler".
// When a "brawler" is closer than 50m, fire artillery.

while (true) {
    // Find the nearest enemy and the distance to it.

    // If the enemy's type is "brawler"
    // AND the distance to it is less than 50 meters,
    // Then say "Fire!" to signal the artillery.

}
```

Overview

You know how to use a logical AND operator from the previous levels. Also, you should know how to check an ogre's type and the distance to it.

Check each ogre's type AND distance before firing!

Brawler Hunt Solution

```
// Don't worry about small and medium-sized ogres.
// Your targets are type "brawler".
// When a "brawler" is closer than 50m, fire artillery.

while (true) {
    // Find the nearest enemy and the distance to it.
    var enemy = hero.findNearestEnemy();
    var distance = hero.distanceTo(enemy);
    // If the enemy's type is "brawler"
    // AND the distance to it is less than 50 meters,
    // Then say "Fire!" to signal the artillery.
    if (enemy.type == "brawler" && distance < 50) {
        hero.say("Fire!");
    }
}
```

#37. Usual Day

Level Overview and Solutions

Intro

When using the AND operator, if the first condition (on the left of the AND) is false, the second condition (on the right) will never have to execute.

You can use this to your advantage!

This code may have an error:

```
var enemy = hero.findNearestEnemy();
// Trying to get enemy.type is an error if enemy is null!
if(enemy.type == "munchkin") {
    hero.attack(enemy);
}
```

This code with AND will not cause an error:

```
var enemy = hero.findNearestEnemy();
// If enemy is null, the && is false
// So enemy.type will not be executed, and won't cause an error.
if(enemy && enemy.type == "munchkin") {
    hero.attack(enemy);
}
```

Default Code

```
// Defeat munchkins, collect coins. Everything as usual.
// Use AND to check existence and type in one statement.

while (true) {
    var enemy = hero.findNearestEnemy();
    // With AND, the type is only checked if enemy exists.
    if (enemy && enemy.type == "munchkin") {
        hero.attack(enemy);
    }
    // Find the nearest item.

    // Collect item if it exists and its type is "coin".

}
```

Overview

The **AND** operator has a short-circuit evaluation feature.

This means that if the first expression before `and` is `false` (or null), then the second expression after the operator isn't executed or evaluated.

We can use it when we need to read a property of an object, but we aren't sure the object exists. For example, when we need to read an enemy's type, first we should be sure that enemy exists or we'll get the error:

```
var enemy = hero.findNearestEnemy();
if (enemy) {
    if (enemy.type == "burl") {
        // Do something
    }
}
```

But we can make it shorter:

```
var enemy = hero.findNearestEnemy();
if (enemy && enemy.type == "burl") {
        // Do something
    }
}
```

We don't have to worry about the reference error if there isn't an enemy, becase the second part isn't evaluated in this case.

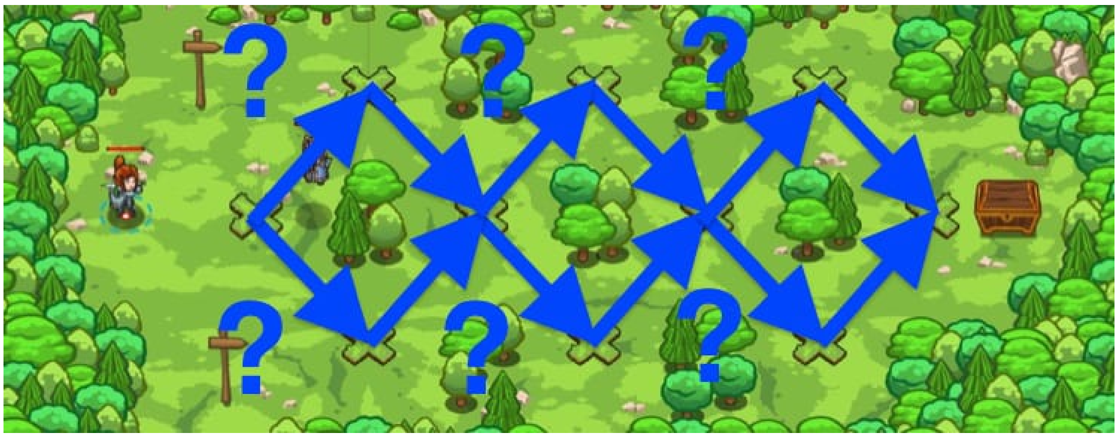
Usual Day Solution

```
// Defeat munchkins, collect coins. Everything as usual.
// Use AND to check existence and type in one statement.

while (true) {
    var enemy = hero.findNearestEnemy();
    // With AND, the type is only checked if enemy exists.
    if (enemy && enemy.type == "munchkin") {
        hero.attack(enemy);
    }
    // Find the nearest item.
    var item = hero.findNearestItem();
    // Collect item if it exists and its type is "coin".
    if (item && item.type == "coin") {
        hero.moveXY(item.pos.x, item.pos.y);
    }
}
```

#38. Logical Path

Level Overview and Solutions

Intro



Deep in the woods, a wizard presents you with a challenge: if you can figure out his logical riddles, he'll share his treasure with you!

Use *boolean operators* like AND, OR, and NOT to determine which paths to take at each fork in the road. Check the guide for tips on how to write the code you want.

Default Code

```
// Get two secret true/false values from the wizard.
hero.moveXY(14, 24);
var secretA = hero.findNearestFriend().getSecretA();
var secretB = hero.findNearestFriend().getSecretB();

// If BOTH secretA and secretB are true, take the high path; otherwise, take the low path.
// Check the guide for notes on how to write logical expressions.
var secretC = secretA && secretB;
if (secretC)
    hero.moveXY(20, 33);
else
    hero.moveXY(20, 15);
hero.moveXY(26, 24);

// If EITHER secretA or secretB is true, take the high path.


// If secretB is NOT true, take the high path.
```

Overview

In this level, you choose paths based on *boolean values* (TRUE or FALSE). At each fork in the road, take the high path if the value is TRUE, or the low path if it's FALSE. The wizard gives you the first two values; the rest have to be calculated using *boolean logic.*

Boolean operators work on TRUE and FALSE values, and give TRUE or FALSE answers.

The AND operator returns TRUE if *both* of its inputs are true:

```
// && means AND.
var a = true && true;   // a = true
var b = true && false;  // b = false
var c = false && true;  // c = false
var d = false && false; // d = false
```

The OR operator returns TRUE if *either* of its inputs are true:

```
// || means OR.
var a = true || true;   // a = true
var b = true || false;  // b = true
var c = false || true;  // c = true
var d = false || false; // d = false
```

The NOT operator works a little differently. It works on one input value, and just gives you the opposite:

```
// ! means NOT.
var a = !true;  // a = false
var b = !false; // b = true
```

As always, you can mix literal TRUE or FALSE values with variables when using boolean logic:

```
var a = !false;      // a = true
var b = a || false;  // b = true
var c = a && b;      // c = true
```

Use all these tools to find your way to the treasure at the end of the path!

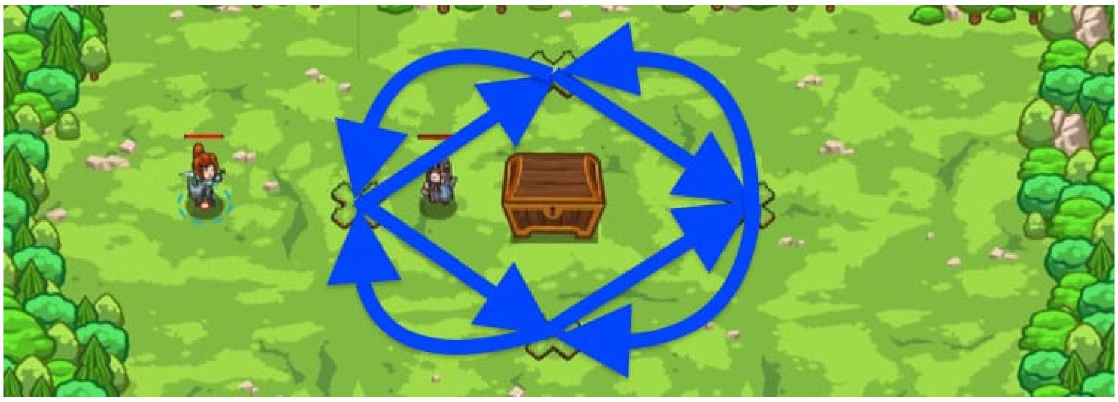
Logical Path Solution

```
// Get two secret true/false values from the wizard.
hero.moveXY(14, 24);
var secretA = hero.findNearestFriend().getSecretA();
var secretB = hero.findNearestFriend().getSecretB();

// If BOTH secretA and secretB are true, take the high path; otherwise, take the low path.
// Check the guide for notes on how to write logical expressions.
var secretC = secretA && secretB;
if (secretC)
    hero.moveXY(20, 33);
else
    hero.moveXY(20, 15);
hero.moveXY(26, 24);

// If EITHER secretA or secretB is true, take the high path.
var secretD = secretA || secretB;
if (secretD)
    hero.moveXY(32, 33);
else
    hero.moveXY(32, 15);
hero.moveXY(38, 24);

// If secretB is NOT true, take the high path.
var secretE = !secretB;
if (secretE)
    hero.moveXY(44, 33);
else
    hero.moveXY(44, 15);
hero.moveXY(50, 24);
```

#39. Logical Circle

Level Overview and Solutions

Intro



Lukacs has led you to another grove with even more treasure to be had! He just needs you to answer a few more of his riddles...

String together chains of boolean values with ANDs and ORs to do more complex logic.

Default Code

```
// Move to the wizard and get their secret values.
hero.moveXY(20, 24);
var secretA = hero.findNearestFriend().getSecretA();
var secretB = hero.findNearestFriend().getSecretB();
var secretC = hero.findNearestFriend().getSecretC();

// If ALL three values are true, take the high path. Otherwise, take the low path.
var secretD = secretA && secretB && secretC;
if (secretD)
    hero.moveXY(30, 33);
else
    hero.moveXY(30, 15);

// If ANY of the three values are true, take the left path. Otherwise, go right.


// If ALL five values are true, take the high path. Otherwise, take the low path.
```

Overview

Like in "Logical Path," you will use *boolean logic* to decide which path to take, but this time, you'll be walking around a large chest full of treasure.

First, you'll take either the high or low path depending on whether the secret value is TRUE or FALSE. Then from there, you'll go either left or right. Then high or low again, and so on and so forth.

Boolean operators like AND and OR can work on more than two values at a time. AND is TRUE if *all* of its values are TRUE:

```
var a = true && true && true;     // a = true
var b = true && false && true;    // b = false
var c = false && true && false;   // c = false
var d = false && false && false; // d = false
var e = true && true && true && true && false && true; # e = false
```

OR is TRUE if *any* of its values are TRUE:

```
var a = true || true || true;    // a = true
var b = true || false || true;   // b = true
var c = false || true || false;  // c = true
var d = false || false || false; // d = false
var e = false || false || false || false || true || false; // e = true
```

Use variables to track of which values are being asked for, and just chain them together to get to Lukacs's treasure!

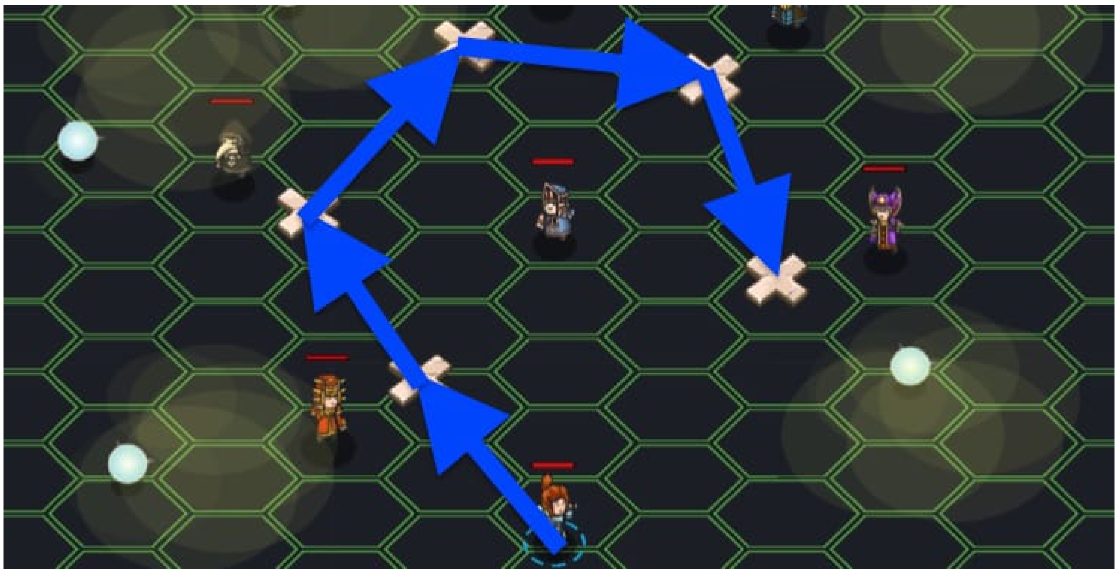
Logical Circle Solution

```
// Move to the wizard and get their secret values.
hero.moveXY(20, 24);
var secretA = hero.findNearestFriend().getSecretA();
var secretB = hero.findNearestFriend().getSecretB();
var secretC = hero.findNearestFriend().getSecretC();

// If ALL three values are true, take the high path. Otherwise, take the low path.
var secretD = secretA && secretB && secretC;
if (secretD)
    hero.moveXY(30, 33);
else
    hero.moveXY(30, 15);

// If ANY of the three values are true, take the left path. Otherwise, go right.
var secretE = secretA || secretB || secretC;
if (secretE)
    hero.moveXY(20, 24);
else
    hero.moveXY(40, 24);

// If ALL five values are true, take the high path. Otherwise, take the low path.
var secretF = secretA && secretB && secretC && secretD && secretE;
if (secretF)
    hero.moveXY(30, 33);
else
    hero.moveXY(30, 15);
```

#40. Logical Conclusion

Level Overview and Solutions

Intro



The gold that Lukacs has given you wasn't his to share! The wizards of the astral plane are none too pleased, and have summoned him to answer for his crimes. It will take your sharpest thinking to answer the wizards' riddles and convince them to free Lukacs and let you return to the forest!

Go to Eszter and get the three secret TRUE/FALSE values you need to start the challenge. Then run the gauntlet of wizards and figure out each one's secret TRUE/FALSE value. Use parentheses to make sure everything is computed in the correct order!

Default Code

```
// Move to Eszter and get three secret values from her.
hero.moveXY(24, 16);
var secretA = hero.findNearestFriend().getSecretA();
var secretB = hero.findNearestFriend().getSecretB();
var secretC = hero.findNearestFriend().getSecretC();

// Say "TRUE" to Tamas if A AND B are true, OR if C is true. Otherwise, say "FALSE."
// Remember to use parentheses to do your logic in the proper order.
var tam = (secretA && secretB) || secretC;
hero.moveXY(19, 26);
hero.say(tam);

// Say "TRUE" to Zsofi if A OR B is true, AND if C is true. Otherwise, say "FALSE."


// Say "TRUE" to Istvan if A OR C is true, AND if B OR C is true. Otherwise, say "FALSE."


// Say "TRUE" to Csilla if A AND B are true, OR if B is true AND C is NOT true. Otherwise, say "FALSE."
```

Overview

In this level you will use advanced boolean logic to decide what to say to each wizard. Like in "Logical Circle," you'll begin with three secret values, and you'll combine them in different ways to solve each puzzle. But in this level, you will need to mix AND, OR, and NOT statements in the correct ways to come up with the correct answers.

When mixing boolean operators, NOT takes precedence over AND, which takes precedence over OR:

```
var a = true && false || true;
//    =     false     || true
//    =             true

var b = true || false && true;
//    = true ||     false
//    =     true

var c = true || !true && true;
//    = true || false && true
//    = true ||     false
//    =     true
```

That's way too hard to read, though! Just use parentheses to make it clear which operators go first:

```
var a = (true && false) || true;
//    =     false       || true
//    =               true

var b = true || (false && true);
//    = true ||      false
//    =     true

var c = true || ((!true) && true);
//    = true || ( false  && true)
//    = true ||        false
//    =     true
```

Using parentheses also allows us to change the order that operators work in:

```
var a = !true || true && false;
//    = false || true && false
//    = false ||    false
//    =     false

var b = !((true || true) && false);
//    = !(     true      && false)
//    = !             false
//    = true
```

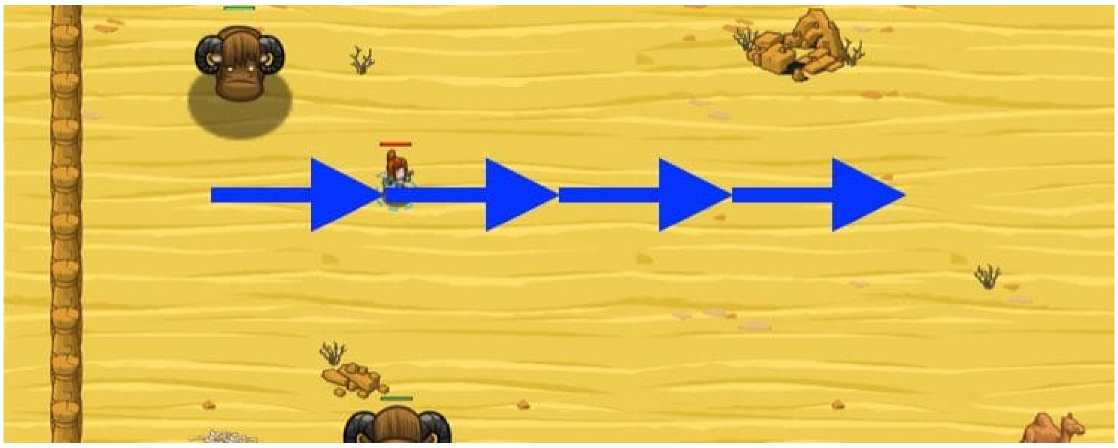
Logical Conclusion Solution

```
// Move to Eszter and get three secret values from her.
hero.moveXY(24, 16);
var secretA = hero.findNearestFriend().getSecretA();
var secretB = hero.findNearestFriend().getSecretB();
var secretC = hero.findNearestFriend().getSecretC();

// Say "TRUE" to Tamas if A AND B are true, OR if C is true. Otherwise, say "FALSE."
// Remember to use parentheses to do your logic in the proper order.
var tam = (secretA && secretB) || secretC;
hero.moveXY(19, 26);
hero.say(tam);

// Say "TRUE" to Zsofi if A OR B is true, AND if C is true. Otherwise, say "FALSE."
var zso = (secretA || secretB) && secretC;
hero.moveXY(26, 36);
hero.say(zso);

// Say "TRUE" to Istvan if A OR C is true, AND if B OR C is true. Otherwise, say "FALSE."
var ist = (secretA || secretC) && (secretB || secretC);
hero.moveXY(37, 34);
hero.say(ist);

// Say "TRUE" to Csilla if A AND B are true, OR if B is true AND C is NOT true. Otherwise, say "FALSE."
var csi = (secretA && secretB) || (secretB && (!secretC));
hero.moveXY(40, 22);
hero.say(csi);
```

#41. The Mighty Sand Yak

Level Overview and Solutions

Intro



`If` a yak gets within 10 meters, dodge to the right by adding 10 to `hero.pos.x`!

```
var x = hero.pos.x;
var y = hero.pos.y;

x = x + 10;
hero.moveXY(x, y);
```

Default Code

```
// Let yaks get close, then move 10m right to dodge.
// Dodge 4 yaks to complete the level.

while(true) {
    // Get hero's current x and y position.
    var x = hero.pos.x;
    var y = hero.pos.y;

    // Find the nearest yak.
    var yak = hero.findNearestEnemy();

    // If the distanceTo the yak is less than 10:
    if (hero.distanceTo(yak) < 10) {
        // To move right, add 10 to hero's x position.

        // Use moveXY(x, y) to move!

    }
}
```

Overview

The `hero.pos` property represents your hero's current position. This `pos` object has two properties, `x` and `y`:

```
var x = hero.pos.x;
var y = hero.pos.y;
```

These are **numbers**, representing the hero's position on the map.

So, if you want to move your hero 10 meters to the right of their current position, that would be:

```
x = x + 10;
```

And `y` would be the same (no movement up or down).

Move to the new coordinates with `hero.moveXY(x, y)`.

You might sometimes see this all written on one line, like:

```
hero.moveXY(hero.pos.x + 10, hero.pos.y)
```

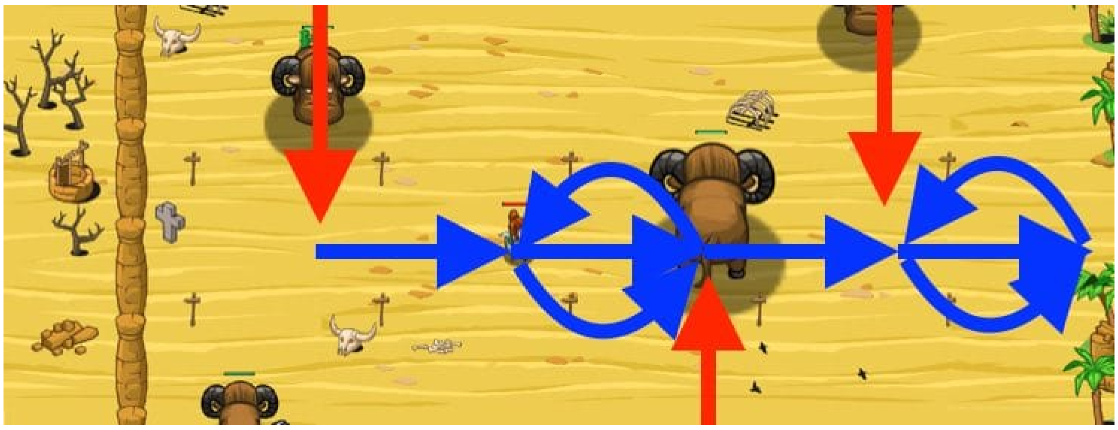
The Mighty Sand Yak Solution

```
// Let yaks get close, then move 10m right to dodge.
// Dodge 4 yaks to complete the level.

while(true) {
    // Get hero's current x and y position.
    var x = hero.pos.x;
    var y = hero.pos.y;

    // Find the nearest yak.
    var yak = hero.findNearestEnemy();

    // If the distanceTo the yak is less than 10:
    if (hero.distanceTo(yak) < 10) {
        // To move right, add 10 to hero's x position.
        x += 10;
        // Use moveXY to move!
        hero.moveXY(x, y);
    }
}
```

#42. Oasis

Level Overview and Solutions

Intro



Move right until you get close to a yak, then move left.

To move left, subtract from `hero.pos.x` .

To move right, add to `hero.pos.x` .

Default Code

```
// Move right to reach the oasis,
// Move left to avoid nearby yaks.
while(true) {
    var x = hero.pos.x;
    var y = hero.pos.y;
    var enemy = hero.findNearestEnemy();
    if (enemy && hero.distanceTo(enemy) < 10) {
        // Subtract 10 from x to move left.

        // Use moveXY to move to the new x, y position.

    } else {
        // Add 10 to x to move right.

        // Use moveXY to move to the new x, y position.

    }
}
```

Overview

In this level, you access your `hero.pos` to move relative to the hero's current position.

If the nearest `"sand-yak"` is less than `10` meters away, move `10` meters to the **left** (subtract from your `x` position) to dodge it.

Otherwise, move `10` meters to the **right** (add to your `x` position) to get closer to the oasis.

Hint: Using a partial move technique like this lets you change what you're doing en route to a position, which is very helpful for responding to changing threats and new information.

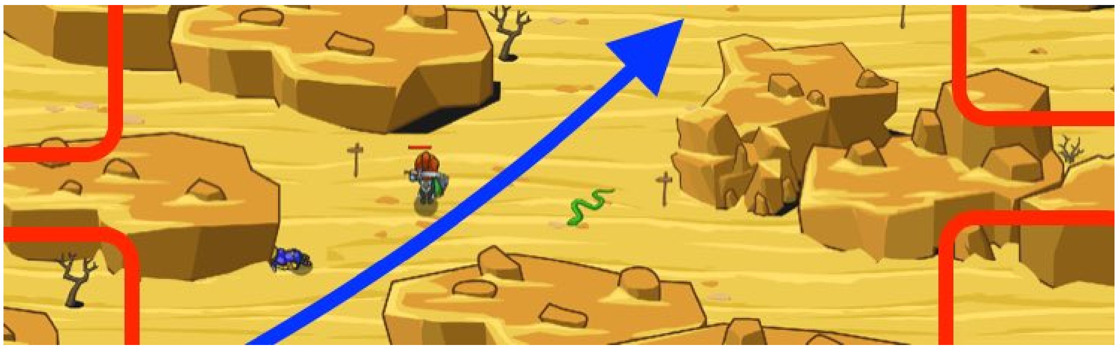
Oasis Solution

```
// Move right to reach the oasis,
// Move left to avoid nearby yaks.
while(true) {
    var x = hero.pos.x;
    var y = hero.pos.y;
    var enemy = hero.findNearestEnemy();
    if (enemy && hero.distanceTo(enemy) < 10) {
        // Subtract 10 from x to move left.
        x -= 10;
        // Use moveXY to move to the new x, y position.
        hero.moveXY(x, y);
    } else {
        // Add 10 to x to move right.
        x += 10;
        // Use moveXY to move to the new x, y position.
        hero.moveXY(x, y);
    }
}
```

#43. Sarven Road

Level Overview and Solutions

Intro



If you see an enemy, attack it! Otherwise, move up and to the right.

You move up by adding to your `pos.y`.

You move right by adding to your `pos.x`.

Default Code

```
// Get to the oasis. Watch out for new enemies: ogre scouts!
// Go up and right by adding to the current X and Y position.

while(true) {
    // If there's an enemy, attack.

    // Else, keep moving up and to the right.
}
```

Overview

You move diagonally up and to the right by adding to both your `x` position and your `y` position.

First `findNearestEnemy`.

`if` there is one, attack it!

`else` (there is no enemy), move to the new `x + 5` and `y + 5` coordinates.

The ogre scouts are tougher than munchkins.

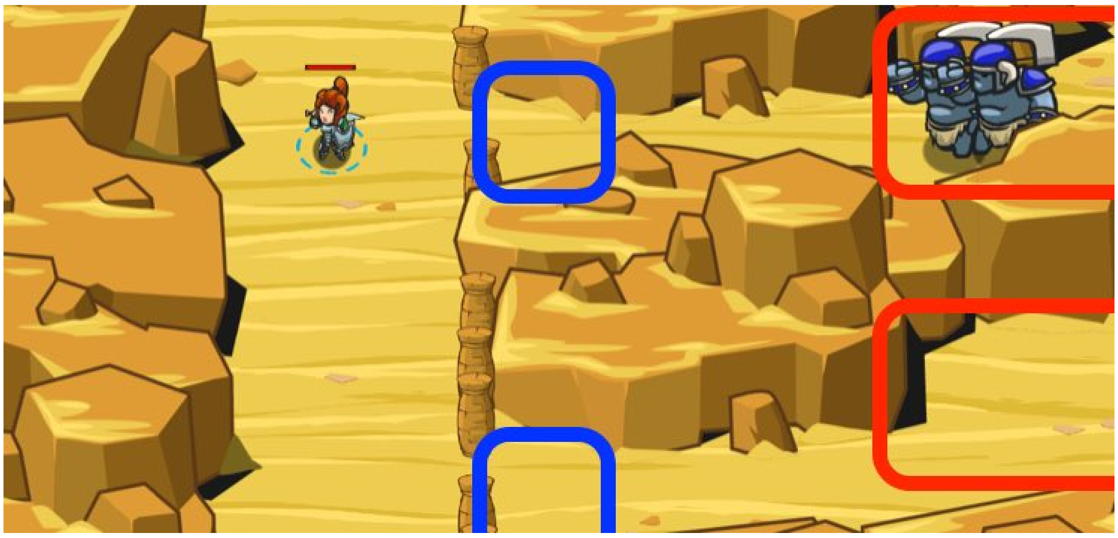
Sarven Road Solution

```
// Get to the oasis. Watch out for new enemies: ogre scouts!
// Go up and right by adding to the current X and Y position.

while(true) {
    // If there's an enemy, attack.
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        hero.attack(enemy);
    }
    // Else, keep moving up and to the right.
    else {
        var xPos = hero.pos.x;
        var yPos = hero.pos.y;
        hero.moveXY(xPos + 5, yPos + 5);
    }
}
```

#44. Sarven Gaps

Level Overview and Solutions

Intro



You can get an enemy's `pos` with `enemy.pos`, then add or subtract from their `x` and `y` coordinates, just like your `hero`.

Remember, subtract from `y` to go down. Subtract from `x` to go left.

Default Code

```
// Get to the Oasis by moving down 10m at a time.
// Build fences 20m to the left of each ogre.

while(true) {
    var enemy = hero.findNearestEnemy();
    if (enemy) {
        // buildXY a "fence" 20 meters to enemy's left.

    } else {
        // moveXY down 10 meters.

    }
}
```

Overview

Use what you learned in the previous desert levels to move **down** relative to your `hero.pos` `10` meters at a time.

If you see ogres, build a fence `20` meters to the **left** of the `enemy.pos` so that they can't get through.

Tip: you can access the `pos` property on your hero, on enemies, on friends, and even on items.

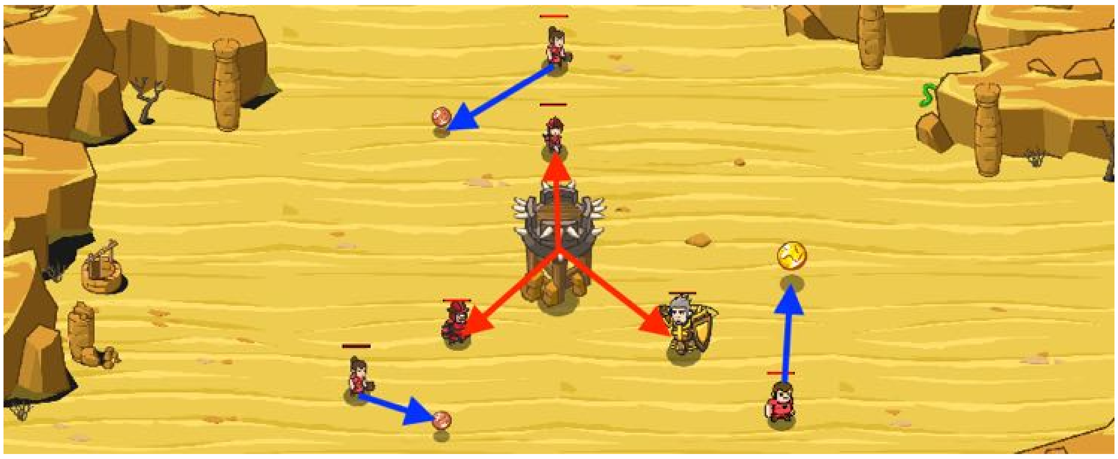
Sarven Gaps Solution

```
// Get to the Oasis by moving down 10m at a time.
// Build fences 20m to the left of each ogre.

while(true) {
    var enemy = hero.findNearestEnemy();
    if (enemy) {
        // buildXY a "fence" 20 meters to enemy's left.
        var x = enemy.pos.x - 20;
        var y = enemy.pos.y;
        hero.buildXY("fence", x, y);
    } else {
        // moveXY down 10 meters.
        var x = hero.pos.x;
        var y = hero.pos.y - 10;
        hero.moveXY(x, y);
    }
}
```

#45. Interception

Level Overview and Solutions

Intro



Protect the peasant by moving to the spot between the tower and your friend.

Calculate the halfway point between `enemy` and `friend` for both `x` and `y`.

First, add their `x` coordinates, then divide by 2:

```
var x = (friend.pos.x + enemy.pos.x) / 2;
```

Then do the same for `y`.

Default Code

```
// Stand between the peasant and the tower.

while(true) {
    var enemy = hero.findNearestEnemy();
    var friend = hero.findNearestFriend();
    // Calculate x by adding friend.pos.x to enemy.pos.x
    // Then divide by 2.
    // Check the guide if you need more help!

    // Now do the same for y

    // Move to the x and y coordinates you calculated.

}
```

Overview

Protect the peasant by standing between the tower and your friend.

In this level you'll need to find the position between two points.

Let's start off with a simple example, in 1D!

Say the tower is at `x: 0`, and the peasant is at `x: 10`. What is the center point between the tower and the peasant?

```
(0 + 10) / 2 == 5
```

Which we know is true because `5` is above `0` and below `10`.

But what if the tower isn't at `0`? Well, it still works! Moving the tower to `6`, we need to find a point between the tower and the peasant.

```
(6 + 10) / 2 == 8
```

This still passes our test, but now what about in both X and Y? Harder to check, but it works similarly:

```
x = (tower.pos.x + peasant.pos.x) / 2
y = (tower.pos.y + peasant.pos.y) / 2
```

Now hop back in and test it out, you'll move similarly to the soldiers who are protecting their peasants!

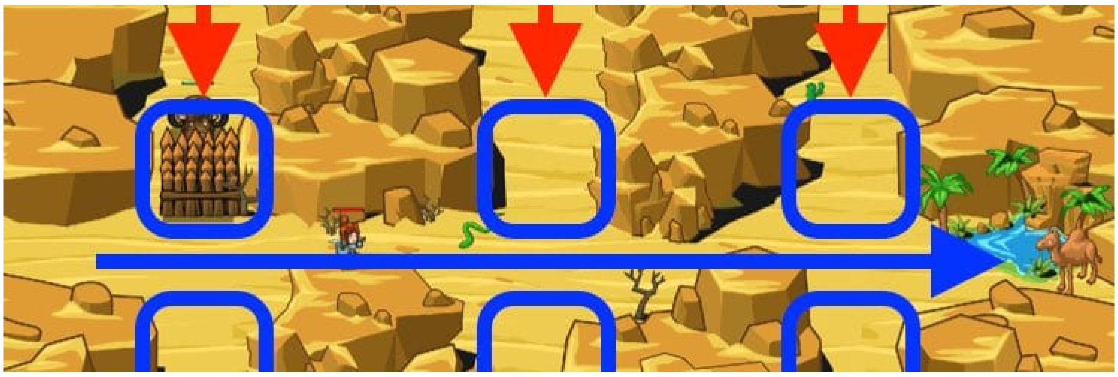
Interception Solution

```
// Stand between the peasant and the tower.

while(true) {
    var enemy = hero.findNearestEnemy();
    var friend = hero.findNearestFriend();
    // Calculate x by adding friend.pos.x to enemy.pos.x
    // Then divide by 2.
    // Check the guide if you need more help!
    var x = (friend.pos.x + enemy.pos.x) / 2;
    // Now do the same for y
    var y = (friend.pos.y + enemy.pos.y) / 2;
    // Move to the x and y coordinates you calculated.
    hero.moveXY(x, y);
}
```

#46. Thunderhooves

Level Overview and Solutions

Intro



Build `"fence"` s to block the yaks coming at you randomly from above and below. Move right towards the oasis if you can't see a yak.

When you see a yak, compare its `pos.y` to your `hero.pos.y` .

If the yak's y position is greater than hero's, the yak is above the hero. Otherwise, the yak is below the hero.

Default Code

```
// Move right, to the oasis.
// Build a "fence" above or below when you see a yak.

while(true) {
    var yak = hero.findNearestEnemy();
    if (yak) {
        // If yak.pos.y is greater than hero.pos.y

            // buildXY a "fence" 10m below the yak.

        // else:

            // buildXY a "fence" 10m above the yak.

    } else {
        // moveXY right 10m towards the oasis.

    }
}
```

Overview

The sand yaks are randomly coming from either the top or the bottom, so you'll have to compare their `y` position with your `y` position to decide whether to build fences above or below them.

If the yak's `pos.y` is greater than your `pos.y` , that means the yak is above, so you should build below the yak. Otherwise, build above the yak.

Hint: Use the greater-than operator `>` on the yak's position's y property.

Hint: Remember, use `buildXY` to build a `"fence"` .

Thunderhooves Solution

```
// Move right, to the oasis.
// Build a "fence" above or below when you see a yak.

while(true) {
    var yak = hero.findNearestEnemy();
    if (yak) {
        // If yak.pos.y is greater than hero.pos.y
        if(yak.pos.y > hero.pos.y) {
            // buildXY a "fence" 10m below the yak.
            hero.buildXY("fence", yak.pos.x, yak.pos.y - 10);
        // else:
        } else {
            // buildXY a "fence" 10m above the yak.
            hero.buildXY("fence", yak.pos.x, yak.pos.y + 10);
        }
    } else {
        // moveXY right 10m towards the oasis.
        hero.moveXY(hero.pos.x + 10, hero.pos.y);
    }
}
```

#47. Kithgard Enchanter

Level Overview and Solutions

Intro

Remember the simpler times, back in Kithgard Dungeon when you were running for your life with nothing but a pair of Simple Boots?

Well, this time you don't even have the boots!

You'll need to enchant your own boots with `moveRight` , `moveUp` , and `moveDown` functions!

Default Code

```
// Define your own simple movement functions.
// Define moveRight
// Note: each function should move the hero 12 meters!
function moveRight() {
    var x = hero.pos.x + 12;
    var y = hero.pos.y;
    hero.moveXY(x,y);
}

// Define moveDown

// Define moveUp

// Now, use those functions!
moveRight();
moveDown();
moveUp();
moveUp();
moveRight();
```

Overview

Functions are an important part of coding.

You've been using functions all along: any time you write code like:

```
hero.attack(enemy);
```

...you are "calling" (or "invoking") a function called `attack` .

The actual code that gets executed when you call `attack` is long and complex. Imagine if you had to write 25 lines of code in your program each time you wanted to swing your sword!

That's the first benefit of functions: they reduce a whole bunch of code down into one line.

Not only does this save you from having to re-type the same code over and over, it also makes your code easier to understand, because it takes what might really be complicated logic ("Ok so I want to attack. Do I have a weapon? Am I close enough to hit with my weapon? How long does it take to use my weapon? Do I hit? Do I cause damage?"), and makes it an easy to understand idea: `attack` .

Now you will not only be calling functions, you will **define** your own functions!

Defining a function has two parts: the **name** and the **body**.

The name is the thing you will use to call the function later, like `attack`.

The body is the code that will be executed when the function is called.

There are also sometimes **arguments** (like the *enemy* in `attack(enemy)`) but we will get into that in future levels.

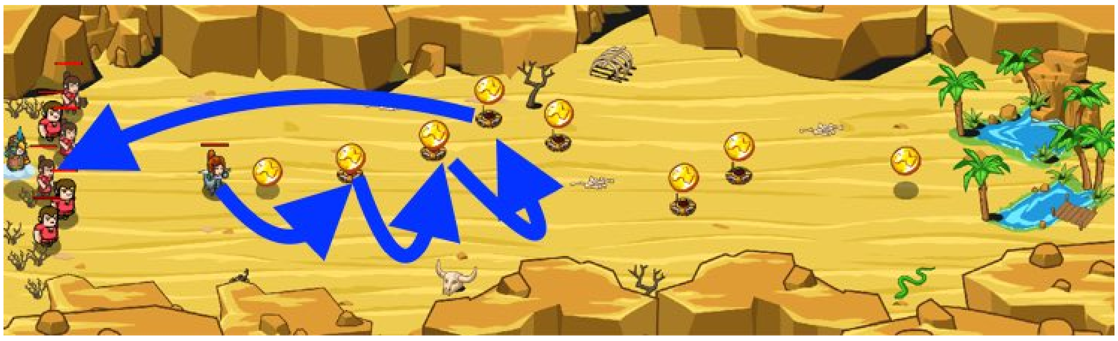
Kithgard Enchanter Solution

```
// Define your own simple movement functions.
// Define moveRight
// Note: each function should move the hero 12 meters!
function moveRight() {
    var x = hero.pos.x + 12;
    var y = hero.pos.y;
    hero.moveXY(x, y);
}

// Define moveDown
function moveDown() {
    var x = hero.pos.x;
    var y = hero.pos.y - 12;
    hero.moveXY(x, y);
}

// Define moveUp
function moveUp() {
    var x = hero.pos.x;
    var y = hero.pos.y + 12;
    hero.moveXY(x, y);
}

// Now, use those functions!
moveRight();
moveDown();
moveUp();
moveUp();
moveRight();
```

#48. Minesweeper

Level Overview and Solutions

Intro



Clear the path for your allies!

Move to each coin, and step on any mines that might be under them.

If your health gets too low, move left 10 meters and say `"Heal please!"` to the Doctor.

Default Code

```
// Lead the peasants and healer through the minefield.

while(true) {
    var coin = hero.findNearestItem();
    var healingThreshold = hero.maxHealth / 2;
    // Check to see if you are critically injured.
    if(hero.health < healingThreshold) {
        // Move left 10m.

        // Ask for a heal.
        hero.say("Can I get a heal?");
    // Else, move to the next coin.
    } else if (coin) {
        hero.moveXY(coin.pos.x, coin.pos.y);
    }
}
```

Overview

Detonate the mines with your own body! The peasants will follow you, but you'll need to run back 10m to the left to ask Doctor Beak for a heal if your health gets too far below your maxHealth.

To trigger a mine, move to the coin on top of it. You can use the `findNearestItem` method to know where to go.

Minesweeper Solution

```
// Lead the peasants and healer through the minefield.

while(true) {
    var coin = hero.findNearestItem();
    var healingThreshold = hero.maxHealth / 2;
    // Check to see if you are critically injured.
    if(hero.health < healingThreshold) {
        // Move left 10m.
        var x = hero.pos.x;
        var y = hero.pos.y;
        hero.moveXY(x - 10, y);
        // Ask for a heal.
        hero.say("Can I get a heal?");
    // Else, move to the next coin.
    } else if (coin) {
        hero.moveXY(coin.pos.x, coin.pos.y);
    }
}
```

#49. Operation 'Killdeer'

Level Overview and Solutions

Intro

That is a big band of ogres!

Use a function with a boolean return value to check if the hero should retreat.

Fight until `shouldRun()` returns TRUE, then run!

Default Code

```
// Lure the ogres into a trap. These ogres are careful.
// They will only follow if the hero is injured.

// This function checks the hero's health
// and returns a Boolean value.
function shouldRun() {
    if (hero.health < hero.maxHealth / 2) {
        return true;
    } else {
        return false;
    }
}

while (true) {
    // Move to the X only if shouldRun() returns true

    hero.moveXY(75, 37);
    // Else, attack!

}
```

Overview

These ogres are wicked smart. They won't chase a hero if `hero.health` is greater than one half of `hero.maxHealth` .

But, if your hero is seriously injured, they might think your retreating is real.

Fight until your health is low enough, then run to the red x-mark.

The `shouldRun()` function will check if you are healthy enough to keep battling.

Use the return value inside of an `if-statement` 's conditional to see if the hero should keep fighting or retreat.

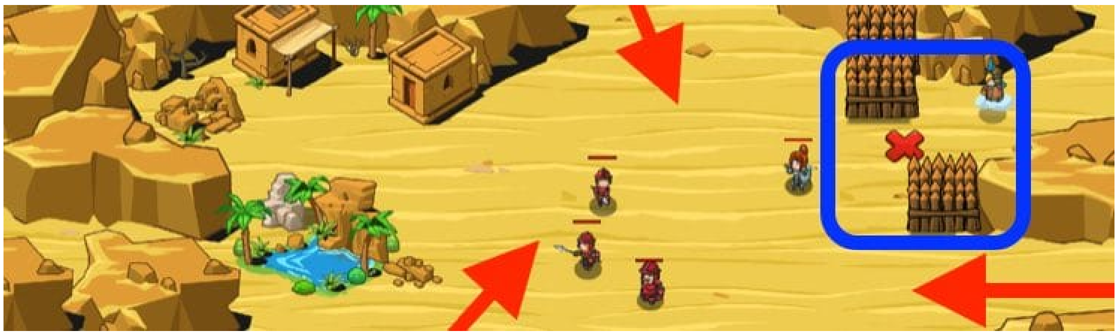
Operation 'Killdeer' Solution

```
// Lure the ogres into a trap. These ogres are careful.
// They will only follow if the hero is injured.

// This function checks the hero's health
// and returns a Boolean value.
function shouldRun() {
    if (hero.health < hero.maxHealth / 2) {
        return true;
    } else {
        return false;
    }
}

while (true) {
    // Move to the X only if shouldRun() returns true
    if(shouldRun()) {
        hero.moveXY(75, 37);
    } else {
        // Else, attack!
        var enemy = hero.findNearestEnemy();
        if(enemy) {
            hero.attack(enemy);
        }
    }
}
```

#50. Medical Attention

Level Overview and Solutions

Intro



Compare your `health` to a fraction of `maxHealth` using the division operator: `/`

Head to the healer and ask for a `"heal"` if you are hurt.

Default Code

```
// Ask the healer for help when you're under one-half health.

while(true) {
    var currentHealth = hero.health;
    var healingThreshold = hero.maxHealth / 2;
    // If your current health is less than the threshold,
    // move to the healing point and say, "heal me".
    // Otherwise, attack. You'll need to fight hard!

}
```

Overview

Sometimes you just have to run back to a medic before you can keep fighting. It happens to all of us. In this level, you compare your `health` to your `maxHealth` divided by 2 with the division operator: `/`

If your `health` is *less than* your `maxHealth` *divided by* `2`, *then* go get healed, *else, if* there is an enemy, *then* keep fighting.

When you go for a heal, make sure you're in position and you say, `"heal me"`.

Medical Attention Solution

```
// Ask the healer for help when you're under one-half health.

while(true) {
    var currentHealth = hero.health;
    var healingThreshold = hero.maxHealth / 2;
    // If your current health is less than the threshold,
    // move to the healing point and say, "heal me".
    // Otherwise, attack. You'll need to fight hard!
    if(hero.health < healingThreshold) {
        hero.moveXY(65, 46);
        hero.say("Heal please!");
    } else {
        var enemy = hero.findNearestEnemy();
        if(enemy) {
            hero.attack(enemy);
        }
    }
}
```

#51. Keeping Time

Level Overview and Solutions

Intro

With `time` you can change your actions based on how much time has passed.

Also, chain if statements together with `else-if`:

```
if(hero.time < 10) {
    // attack
} else if (hero.time < 30) {
    // collect coins
}
```

Default Code

```
// Use your new skill to choose what to do: hero.time

while(true) {
    // If it's the first 10 seconds, attack.
    if (hero.time < 10) {

    }
    // Else, if it's the first 35 seconds, collect coins.
    else if (hero.time < 35) {

    }
    // After 35 seconds, attack again!
    else {

    }
}
```

Overview

You can use your watch's `time` function to determine how much time has passed in the world since the last time you pressed "Run".

In this level, for the first 10 seconds, you should help fight the ogres, like this:

```
if(hero.time < 10) {
    // attack
}
```

Next, we use an **else-if** statement to chain `if` statements:

```
if(hero.time < 10) {
    // attack
} else if (hero.time < 30) {
    // collect coins
}
```

The second `if` statement says to collect coins if `time` is less-than 30 seconds. But this `if` statement is connected to the first `if` statement with an `else` clause, so the coin collecting will only happen when the first if statement is **false**, *and* the second if statement is **true**.

A final `else` clause tells you to help your allies fight the ogres when the first two if statements are false.

Hint: If you're having trouble surviving the final fight, have your hero retreat to safety when your health is low!

Hint: You may also want to avoid attacking if the `enemy.type` is `"palisade"`.

Keeping Time Solution

```
// Use your new skill to choose what to do: hero.time

while(true) {
    // If it's the first 10 seconds, attack.
    if (hero.time < 10) {
        var enemy = hero.findNearestEnemy();
        if(enemy) {
            hero.attack(enemy);
        }
    }
    // Else, if it's the first 35 seconds, collect coins.
    else if (hero.time < 35) {
        var coin = hero.findNearestItem();
        if(coin) {
            hero.moveXY(coin.pos.x, coin.pos.y);
        }
    }
    // After 35 seconds, attack again!
    else {
        var enemy = hero.findNearestEnemy();
        if(enemy) {
            hero.attack(enemy);
        }
    }
}
```

#52. While Ogres Were Sleeping

Level Overview and Solutions

Intro

The ogres are sleeping, so proceed carefully:

First, attack only if `enemy.team` is `"ogres"` AND `enemy.health` is less than `10`.

Second, collect coins only if `coin.value` is less than `5` AND `hero.distanceTo(coin)` is less than `7`.

Third, attack only if `enemy.health` is less than `10` AND `enemy.type` is `"skeleton"`.

Default Code

```
// Enemies are sleeping. It's the perfect time for sabotage!

var points = [{x: 21, y: 8}, {x: 33, y: 8}, {x: 45, y: 8},
    {x: 57, y: 8}, {x: 68, y: 8}, {x: 68, y: 18},
    {x: 68, y: 28}, {x: 68, y: 38}, {x: 68, y: 48},
    {x: 68, y: 58}, {x: 56, y: 58}, {x: 44, y: 58},
    {x: 32, y: 58}, {x: 20, y: 58}, {x: 10, y: 60}];

var pointIndex = 0;

while (pointIndex < points.length) {
    var point = points[pointIndex];
    hero.moveXY(point.x, point.y);
    var enemy = hero.findNearestEnemy();
    var coin = hero.findNearestItem();
    // Attack only if the enemy.team is "ogres"
    // AND the enemy's health is less than 10

    // Collect a coin if coin.value is less than 5
    // AND its distance is less than 7

    // Attack only if the enemy.health is less than 10
    // AND the enemy's type is "skeleton".

    pointIndex++;
}
```

Overview

Ogres are sleeping in the middle of the desert. A perfect time to strike!

Navigate around their camp and defeat choice ogres and loot certain coins.

Remember that the `<`, `>`, `<=`, and `>=` operators just compare two elements and return a boolean value of `true` or `false`.

```
var enemy = hero.findNearestEnemy();
// Read it like it's written:
// IF the distance is less than 10m AND the enemy's health is less than 10!
if(hero.distanceTo(enemy) < 10 && enemy.health < 10) {
    // Defeat the weak enemy!
}
```

While Ogres Were Sleeping Solution

```javascript
// Enemies are sleeping. It's the perfect time for sabotage!

var points = [{x: 21, y: 8}, {x: 33, y: 8}, {x: 45, y: 8},
    {x: 57, y: 8}, {x: 68, y: 8}, {x: 68, y: 18},
    {x: 68, y: 28}, {x: 68, y: 38}, {x: 68, y: 48},
    {x: 68, y: 58}, {x: 56, y: 58}, {x: 44, y: 58},
    {x: 32, y: 58}, {x: 20, y: 58}, {x: 10, y: 60}];

var pointIndex = 0;

while (pointIndex < points.length) {
    var point = points[pointIndex];
    hero.moveXY(point.x, point.y);
    var enemy = hero.findNearestEnemy();
    var coin = hero.findNearestItem();
    // Attack only if the enemy.team is "ogres"
    // AND the enemy's health is less than 10
    if (enemy.team == "ogres" && enemy.health < 10) {
        hero.attack(enemy);
    }
    // Collect a coin if coin.value is less than 5
    // AND its distance is less than 7
    else if (coin.value < 5 && hero.distanceTo(coin) < 5) {
        hero.moveXY(coin.pos.x, coin.pos.y);
    }
    // Attack only if the enemy.health is less than 10
    // AND the enemy's type is "skeleton".
    else if (enemy.health < 10 && enemy.type == "skeleton") {
        hero.attack(enemy);
    }
    pointIndex++;
}
```

#53. Crux of the Desert

Level Overview and Solutions

Intro

Ogres are attacking from diagonal directions.

First, figure out if the enemy `isAbove` or `isBelow`.

Then figure out if the enemy `isLeft` or `isRight`.

Finally, combine above conditions to build `"fire-trap"` s at the correct X mark.

Default Code

```
// Figure out which direction the ogres are coming from.

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        // Left: enemy.pos.x is less than hero.pos.x
        var isLeft = hero.pos.x  > enemy.pos.x;
        // Above: enemy.pos.y is greater than hero.pos.y
        var isAbove = hero.pos.y < enemy.pos.y;
        // Right: enemy.pos.x is greater than hero.pos.x

        // Below: enemy.pos.y is less than hero.pos.y

        // If enemy isAbove and isLeft:
        // buildXY() a "fire-trap" at the X mark.
        if(isLeft && isAbove) {
            hero.buildXY("fire-trap", 40 - 20, 34 + 17);
        }
        // If enemy isAbove and isRight:
        // buildXY() a "fire-trap" at the X mark.

        // If enemy isBelow and isLeft:
        // buildXY() a "fire-trap" at the X mark.

        // If enemy isBelow and isRight:
        // buildXY() a "fire-trap" at the X mark.

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

Overview

In this level, ogres will be assailing from diagonal directions!

Enemies to the **LEFT** will have a `pos.x` that is less than the hero's `pos.x`.

Enemies to the **RIGHT** will have a `pos.x` that is greater th an the hero's `pos.x'.

Enemeis **ABOVE** have a `pos.y` that is greater than the hero's `pos.y`.

Enemies **BELOW** have a `pos.y` that is less than the hero's `pos.y`.

To check if an enemy is Above and Left, the code looks like so:

```
var enemy = hero.findNearestEnemy();
if(enemy) {
    if(enemy.pos.x < hero.pos.x && enemy.pos.y > hero.pos.y) {
        // Build a fire-trap on the NW mark.
    }
}
```

But there is a cleaner way of writing this:

```
var enemy = hero.findNearestEnemy();
if(enemy) {
    var isAbove = enemy.pos.y > hero.pos.y;
    var isLeft = enemy.pos.x < hero.pos.x;
    if(isAbove && isLeft) {
        // Build a fire-trap on the NW mark.
    }
}
```

Not only is this easier to repeat across other directions, but it is easier to read!

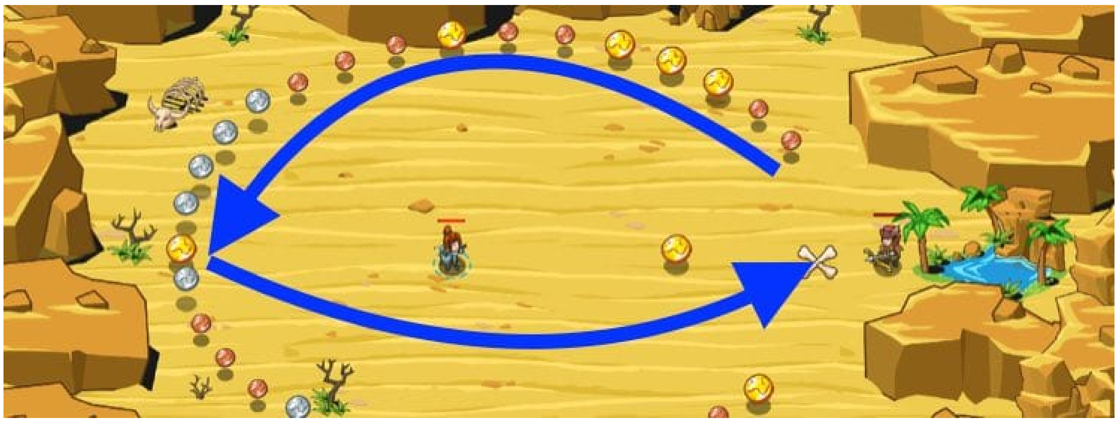
Crux of the Desert Solution

```
// Figure out which direction the ogres are coming from.

while(true) {
    var enemy = hero.findNearestEnemy();
    if(enemy) {
        // Left: enemy.pos.x is less than hero.pos.x
        var isLeft = hero.pos.x  > enemy.pos.x;
        // Above: enemy.pos.y is greater than hero.pos.y
        var isAbove = hero.pos.y < enemy.pos.y;
        // Right: enemy.pos.x is greater than hero.pos.x
        var isRight = hero.pos.x  < enemy.pos.x;
        // Below: enemy.pos.y is less than hero.pos.y
        var isBelow = hero.pos.y > enemy.pos.y;
        // If enemy isAbove and isLeft:
        // buildXY() a "fire-trap" at the X mark.
        if(isLeft && isAbove) {
            hero.buildXY("fire-trap", 40 - 20, 34 + 17);
        }
        // If enemy isAbove and isRight:
        // buildXY() a "fire-trap" at the X mark.
        if(isRight && isAbove) {
            hero.buildXY("fire-trap", 40 + 20, 34 + 17);
        }
        // If enemy isBelow and isLeft:
        // buildXY() a "fire-trap" at the X mark.
        if(isLeft && isBelow) {
            hero.buildXY("fire-trap", 40 - 20, 34 - 17);
        }
        // If enemy isBelow and isRight:
        // buildXY() a "fire-trap" at the X mark.
        if(isRight && isBelow) {
            hero.buildXY("fire-trap", 40 + 20, 34 - 17);
        }
        hero.moveXY(40, 34);
    } else {
        hero.moveXY(40, 34);
    }
}
```

#54. Hoarding Gold

Level Overview and Solutions

Intro



Use a `break` statement to stop a loop before it would normally end.

Default Code

```
// Collect 25 gold, and then tell Naria the total.
// Use break to stop collecting when totalGold >= 25.

var totalGold = 0;
while(true) {
    var coin = hero.findNearestItem();
    if(coin) {
        // Pick up the coin.

        // Add the coin's value to totalGold.
        // Get its value with:  coin.value

    }
    if (totalGold >= 25) {
        // This breaks out of the loop to run code at the bottom.
        // The loop ends, code after the loop will run.
        break;
    }
}

// Done collecting gold!
hero.moveXY(58, 33);
// Go to Naria and say how much gold you collected.
```

Overview

In this level you learn how to `break` out of a loop manually.

When you `break` from a loop, it means the loop immediately stops executing, and your program moves on to whatever code comes after your loop.

The first thing to do is move to pick up the coin you found using `moveXY` and the coin's `pos` property.

Then, be sure to add the value of the coin you picked up to your `totalGold` counter using the coin's `value` property:

```
// add like this:
totalGold = totalGold + coin.value

// or like this:
totalGold += coin.value
```

The sample code shows you how to `break` out of the loop, if your `totalGold` is greater than or equal to 25.

By the way, to tell Naria how much gold you have, you can just `say(totalGold)` , but if you want to get fancy, you can use the string concatenation operator like this:

```
hero.say("Hi Naria, I collected " + totalGold + " gold!")
```

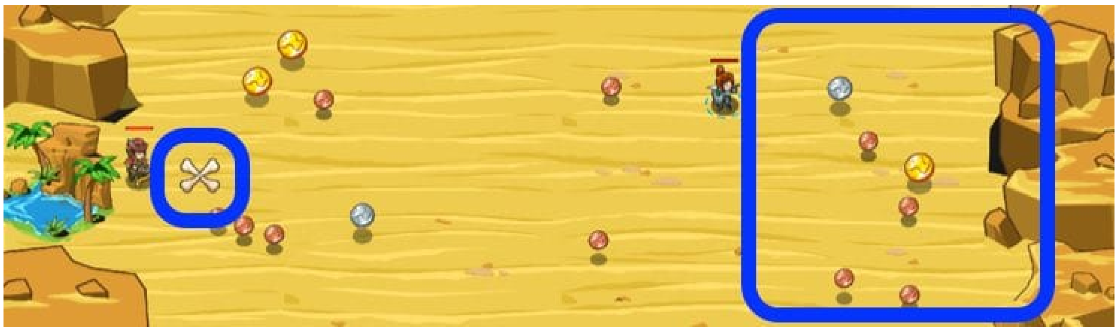
Hoarding Gold Solution

```
// Collect 25 gold, and then tell Naria the total.
// Use break to stop collecting when totalGold >= 25.

var totalGold = 0;
while(true) {
    var coin = hero.findNearestItem();
    if(coin) {
        // Pick up the coin.
        hero.moveXY(coin.pos.x, coin.pos.y);
        // Add the coin's value to totalGold.
        // Get its value with:  coin.value
        totalGold += coin.value;
    }
    if (totalGold >= 25) {
        // This breaks out of the loop to run code at the bottom.
        // The loop ends, code after the loop will run.
        break;
    }
}

// Done collecting gold!
hero.moveXY(58, 33);
// Go to Naria and say how much gold you collected.
hero.say(totalGold);
```

#55. Decoy Drill

Level Overview and Solutions

Intro



Use the `hero.gold` property to build a `"decoy"` when you've collected 25 gold.

```
if(hero.gold >= 25) {
    hero.buildXY("decoy", x, y)
}
```

Default Code

```
// We are field testing a new battle unit: the decoy.
// Build 4 decoys, then report the total to Naria.

var decoysBuilt = 0;
while(true) {
    var coin = hero.findNearestItem();
    if(coin) {
        // Collect the coin!

    }
    // Each decoy costs 25 gold.
    // If hero.gold is greater than or equal to 25:

        // buildXY a "decoy"

        // Add 1 to the decoysBuilt count.


    if(decoysBuilt == 4) {
        // Break out of the loop when you have built 4.

    }
}

hero.say("Done building decoys!");
hero.moveXY(14, 36);
// Say how many decoys you built.
```

Overview

Inside the `while-true loop`, you need to do three things:

1. Collect coins.
2. If `hero.gold` is greater than or equal to 25, Use `buildXY` to build a `"decoy"` and increment `decoysBuilt` by 1

You can check your gold like this:

```
if(hero.gold >= 25) {
    hero.buildXY("decoy", x, y);
}
```

1. If `decoysBuilt` is greater than or equal to 4, `break` out of the loop.

Then, after (outside of) the `while-true loop`, you should `moveXY` to the X mark and use `say` to tell Naria how many `decoysBuilt`

Tip: `buildXY` your decoys at x `hero.pos.x - 5` and y `hero.pos.y`. This will send the decoys to the left, toward Naria.

Decoy Drill Solution

```
// We are field testing a new battle unit: the decoy.
// Build 4 decoys, then report the total to Naria.

var decoysBuilt = 0;
while(true) {
    var coin = hero.findNearestItem();
    if(coin) {
        // Collect the coin!
        hero.moveXY(coin.pos.x, coin.pos.y);
    }
    // Each decoy costs 25 gold.
    // If hero.gold is greater than or equal to 25:
    if(hero.gold >= 25) {
        // buildXY a "decoy"
        hero.buildXY("decoy", hero.pos.x, hero.pos.y);
        // Add 1 to the decoysBuilt count.
        decoysBuilt += 1;
    }
    if(decoysBuilt == 4) {
        // Break out of the loop when you have built 4.
        break;
    }
}

hero.say("Done building decoys!");
hero.moveXY(14, 36);
// Say how many decoys you built.
hero.say(decoysBuilt);
```

#56. Continuous Alchemy

Level Overview and Solutions

Intro

Use the `continue` statement to stop the current iteration of a loop, and start over at the beginning of the next one.

```
while(true) {
    if(!enemy) {
        continue;
    }

    hero.say("I see an enemy!");
}
```

Default Code

```
// Race munchkins to the water distilled by Omarn Brewstone!
// Use the continue statement to avoid poison.
while(true) {
    var enemy = hero.findNearestEnemy();
    var item = hero.findNearestItem();

    // If there is no enemy, continue out of the loop.
    if(!enemy) {
        continue;
    }
    // If there is no item, ask for a potion then continue:
    if(!item) {
        hero.say("Give me a drink!");
        continue;
    }
    // If the item.type "poison", continue out of the loop.

    // At this point, the potion must be a bottle of water
    // so moveXY to the potion, then back to the start!

}
```

Overview

This level will teach you the uses of the `continue` statement. When the program points to a `continue` statement, the rest of the code in the current iteration of the loop is discarded and the next cycle of the loop begins.

Omarn Brewstone is an important man. It is important to check that there is an enemy to ambush, and, there are no other items on the field.

Be sure to check the `item.type` as to not accidentally drink some poison!

Continue can be used to skip blocks of code until a condition is met:

```
while(true) {
    if(!enemy) {
        continue;
    }

    hero.say("I see an enemy!");
}
```

Do not let the Munchkin reach the Water! The desert dehydrated Ogres will become powerful if they're given the opportunity to quench their thirst.

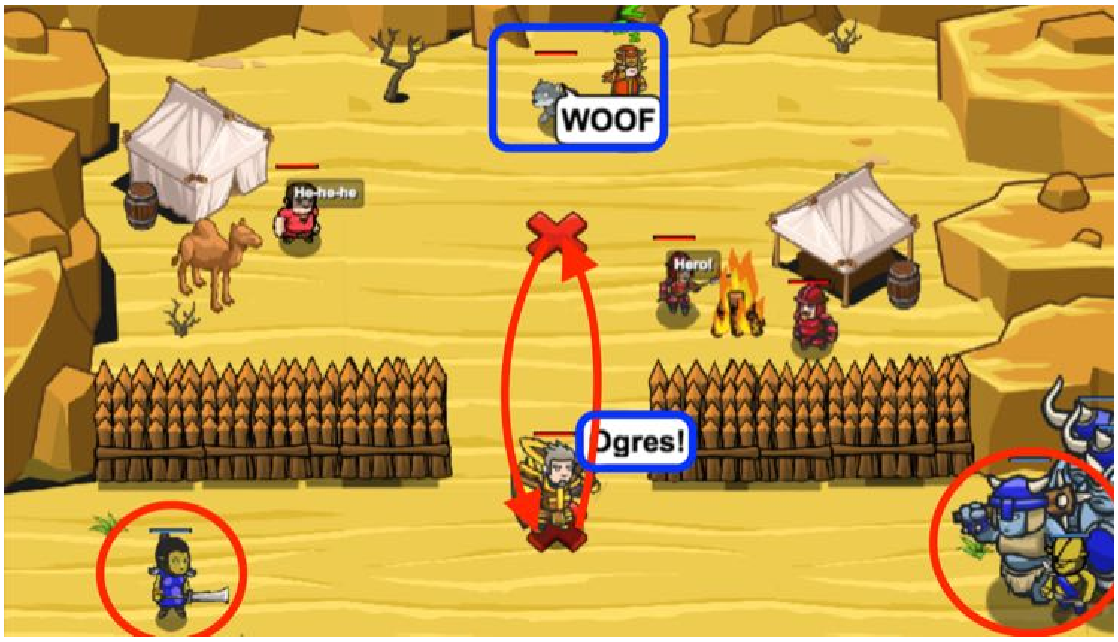
Continuous Alchemy Solution

```
// Race munchkins to the water distilled by Omarn Brewstone!
// Use the continue statement to avoid poison.
while(true) {
    var enemy = hero.findNearestEnemy();
    var item = hero.findNearestItem();

    // If there is no enemy, continue out of the loop.
    if(!enemy) {
        continue;
    }
    // If there is no item, ask for a potion then continue:
    if(!item) {
        hero.say("Give me a drink!");
        continue;
    }
    // If the item.type "poison", continue out of the loop.
    if(item.type == "poison") {
        continue;
    }
    // At this point, the potion must be a bottle of water
    // so moveXY to the potion, then back to the start!
    hero.moveXY(44, 35);
    hero.moveXY(34, 47);
}
```

#57. Chain of Command

Level Overview and Solutions

Intro



Event handler functions are given information about the `event` that triggered them.

Inside the pet's `"hear"` event handler, you can check who the pet heard with `event.speaker`:

This way, you can have your pet only respond when the hero speaks:

```
function onHear(event) {
    // Who said that?
    if (event.speaker === hero) {
        // It's the hero!
        pet.say("It's the Master!");
    }
}
pet.on("hear", onHear);
```

Default Code

```
// Only your pet can wake the wizard up.

function onHear(event) {
    // "hear" events set the event.speaker property.
    // Check if the pet has heard the hero:
    if (event.speaker == hero) {
        pet.say("WOOF");
    }
}

// Assign the event handler for "hear" event.
pet.on("hear", onHear);

while(true) {
    var enemy = hero.findNearestEnemy();
    // If there is an enemy:

        // Use hero.say() to alert your pet

        // Move to the X in the camp.

        // Then return to the X outside the camp.

}
```

Overview

Your pet will only respond to `hero.say()`.

If there are ogres, your hero should:

1. `say()` something to alert the pet.
2. Run to the X inside the camp.
3. Run back to the X outside the camp.

Chain of Command Solution

```
// Only your pet can wake the wizard up.

function onHear(event) {
    // "hear" events set the event.speaker property.
    // Check if the pet has heard the hero:
    if (event.speaker == hero) {
        pet.say("WOOF");
    }
}

// Assign the event handler for "hear" event.
pet.on("hear", onHear);

while(true) {
    var enemy = hero.findNearestEnemy();
    // If there is an enemy:
    if (enemy) {
        // Use hero.say() to alert your pet
        hero.say("Ogres!");
        // Move to the X in the camp.
        hero.moveXY(30, 33);
        // Then return to the X outside the camp.
        hero.moveXY(30, 15);
    }
}
```

#58. Pet Translator

Level Overview and Solutions

Intro

The Sdrawkcab Mercenaries are providing intel on an incoming ogre attack!

Use your pet to decipher what they say using the `event.message` property.

Default Code

```
// Your pet should translate commands.

function onHear(event) {
    // The message the pet heard is in event.message
    var message = event.message;
    // If the message is "North":
    if (message == "North") {
        // The pet says "Htron".
        pet.say("Htron");
    }
    // If the message is "South":
    if (message == "South") {
        // The pet says "Htuos".

    }
    // If the message is "East":

        // The pet says "Tsae".

}

// Assign the event handler.
pet.on("hear", onHear);

while (true) {
    var enemy = hero.findNearestEnemy();
    // Don't attack Brawlers.
    if (enemy && enemy.type != "brawler") {
        hero.attack(enemy);
    }
}
```

Overview

Our scouts have seen several groups of brawlers near the camp. Luckily, we have several hired cannons. Unluckily, the artillerymen don't understand our language.

While your hero is fighting, your pet should translate commands for the mercenaries.

The event handler parameter `event` contains the property `message`.

When a `"hear"` event is triggered, the handler function can access the message the pet heard using `event.message`.

```
function repeat(event) {
    // The pet repeats the heard message.
    pet.say(event.message);
}
```

Pet Translator Solution

```javascript
// Your pet should translate commands.

function onHear(event) {
    // The message the pet heard is in event.message
    var message = event.message;
    // If the message is "North":
    if (message == "North") {
        // The pet says "Htron".
        pet.say("Htron");
    }
    // If the message is "South":
    if (message == "South") {
        // The pet says "Htuos".
        pet.say("Htuos");
    }
    // If the message is "East":
    if (message == "East") {
        // The pet says "Tsae".
        pet.say("Tsae");
    }
}

// Assign the event handler.
pet.on("hear", onHear);

while (true) {
    var enemy = hero.findNearestEnemy();
    // Don't attack Brawlers.
    if (enemy && enemy.type != "brawler") {
        hero.attack(enemy);
    }
}
```

#59. Alchemic Power

Level Overview and Solutions

Intro

Alchemic potions are our secret advantage. While your hero is fighting, your pet should wait for the command `"Fetch"` from the alchemist and deliver a potion to the hero when it is heard.

Use the event handler parameter `event` to get the message that was said. It's contained in the property `event.message`. When the message isn't "Fetch" your pet should return to the red mark. Use `pet.fetch(item)` to take and bring an item to the hero.

Default Code

```
// Wait for alchemist's commands to fetch potions.

// The event handler for the pet's event "hear".
function onHear(event) {
    // Find the nearest potion.
    var potion = pet.findNearestByType("potion");
    // If the event's message is "Fetch"

        // Have the pet fetch the potion.

    // Else (for any other messages):

        // Return the pet to the red mark.

}

pet.on("hear", onHear);

// You don't have to change the code below.
while(true) {
    var enemy = hero.findNearest(hero.findEnemies());
    if (enemy) {
        hero.attack(enemy);
    }
    else {
        hero.moveXY(40, 34);
    }
}
```

Overview

Events include data about what has occured to cause the Event Handler to fire.

For the `"hear"` event, the first parameter (usually named `event`) contains valuable information about who the `speaker` was and what `message` they said.

For example:

```
function onHear(event) {
    var who = event.speaker; // This will be Omarn
    var what = event.message; // This will be "Drink this!"
    pet.say(who + " said " + what); // Omarn said Drink this!
    pet.say("Squawk!") // Squawk!
}
pet.on("hear", onHear);

// Then Omarn says: "Drink this!"
```

Use this to listen for the correct message, and have your pet fetch potions when the time is right!

Alchemic Power Solution

```
// Wait for alchemist's commands to fetch potions.

// The event handler for the pet's event "hear".
function onHear(event) {
    // Find the nearest potion.
    var potion = pet.findNearestByType("potion");
    // If the event's message is "Fetch"
    if (event.message == "Fetch") {
        // Have the pet fetch the potion.
        pet.fetch(potion);
    // Else (for any other messages):
    } else {
        // Return the pet to the red mark.
        pet.moveXY(54, 34);
    }
}

pet.on("hear", onHear);

// You don't have to change the code below.
while(true) {
    var enemy = hero.findNearest(hero.findEnemies());
    if (enemy) {
        hero.attack(enemy);
    }
    else {
        hero.moveXY(40, 34);
    }
}
```

#60. Pet Engineer

Level Overview and Solutions

Intro

Move your pet to the left button (near the robot) when it hears the `archer` call for help.

Move your pet to the right button (near the cannon) when it hears the `soldier` call for help.

Use the `"hear"` event handler's `event.speaker` parameter to recognize who is calling for help!

Default Code

```
// Move your pet to the left or right button as needed.

function onHear(event) {
    // Find the guards to listen to.
    var archer = pet.findNearestByType("archer");
    var soldier = pet.findNearestByType("soldier");
    // If event.speaker is the archer:

        // Move to the left button.

    // If event.speaker is the soldier:

        // Move to the right button.

}

pet.on("hear", onHear);

// You don't have to change the code below.
// Your hero should protect the bottom right passage.
while(true) {
    var enemy = hero.findNearestEnemy();
    if (enemy) {
        hero.attack(enemy);
    }
}
```

Overview

You can check who `event.speaker` is like this:

```
var archer = pet.findNearestByType("archer");
if(event.speaker == archer) {
    // It's the archer
}
```

The left button is located at x `32` , y `30` .

The right button is located at x `48` , y `30` .

Pet Engineer Solution

```
// Move your pet to the left or right button as needed.

function onHear(event) {
    // Find the guards to listen to.
    var archer = pet.findNearestByType("archer");
    var soldier = pet.findNearestByType("soldier");
    // If event.speaker is the archer:
    if (event.speaker == archer) {
        // Move to the left button.
        pet.moveXY(32, 30);
    }
    // If event.speaker is the soldier:
    else if (event.speaker == soldier) {
        // Move to the right button.
        pet.moveXY(48, 30);
    }
}

pet.on("hear", onHear);

// You don't have to change the code below.
// Your hero should protect the bottom right passage.
while(true) {
    var enemy = hero.findNearestEnemy();
    if (enemy) {
        hero.attack(enemy);
    }
}
```

#61. Pet Adjutant

Level Overview and Solutions

Intro

The hero needs to survive 50 seconds before wizards can teleport you to safety.

The pet can react to different things it hears using the `event.message` property.

Move the pet to the bottom X when the hero says `"Fire"` .

Move the pet to the top X when the hero says `"Heal"` .

```
function onHear(event) {
    if (event.message == "Fire") {
        // Fire the cannons.
    }
}
```

You don't have to change the code inside `while-loop` , but you can if you want to experiment with advanced strategies.

Default Code

```
// Your pet can help you survive until you can escape.

function onHear(event) {
    // event.message contains the text that was heard.
    // If somebody said "Fire":
    if (event.message == "Fire") {
        // Move to the bottom X mark with pet.moveXY()

        // Say something with pet.say()

    }
    // If somebody said "Heal":
    else if (event.message == "Heal") {
        // Move to the top X mark with pet.moveXY()

        // Say something with pet.say()

    }
}

pet.on("hear", onHear);

// You don't have to change the code below.
while(true) {
    var enemy = hero.findNearestEnemy();
    if (enemy) {
        // If an enemy is too strong.
        if (enemy.type == "brawler") {
            hero.say("Fire");
        } else {
            hero.attack(enemy);
        }
    } else {
        // If your hero needs healing.
        if (hero.health < hero.maxHealth / 2) {
            hero.say("Heal");
        }
    }
}
```

Overview

Coming soon!

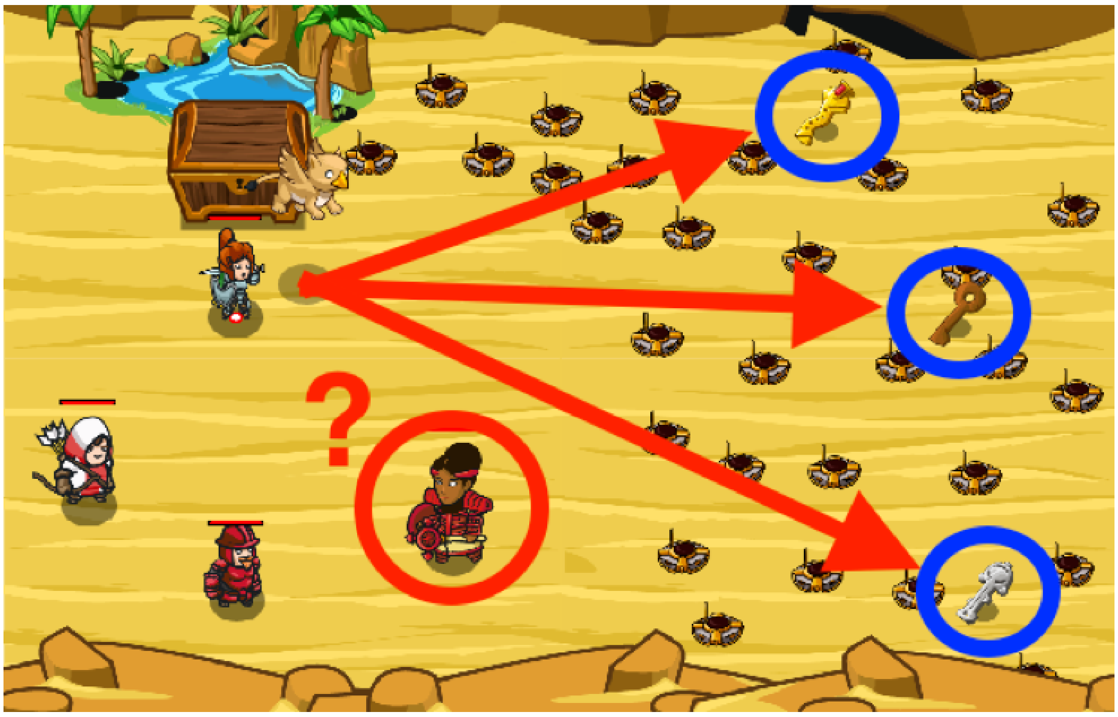
Pet Adjutant Solution

```
// Your pet can help you survive until you can escape.

function onHear(event) {
    // event.message contains the text that was heard.
    // If somebody said "Fire":
    if (event.message == "Fire") {
        // Move to the bottom X mark with pet.moveXY()
        pet.moveXY(64, 16);
        // Say something with pet.say()
        pet.say("MEAW");
    }
    // If somebody said "Heal":
    else if (event.message == "Heal") {
        // Move to the top X mark with pet.moveXY()
        pet.moveXY(64, 52);
        // Say something with pet.say()
        pet.say("MEAW");
    }
}

pet.on("hear", onHear);

// You don't have to change the code below.
while(true) {
    var enemy = hero.findNearestEnemy();
    if (enemy) {
        // If an enemy is too strong.
        if (enemy.type == "brawler") {
            hero.say("Fire");
        } else {
            hero.attack(enemy);
        }
    } else {
        // If your hero needs healing.
        if (hero.health < hero.maxHealth / 2) {
            hero.say("Heal");
        }
    }
}
```

#62. Dangerous Key

Level Overview and Solutions

Intro



Use the correct key and you will get as much gold as you can take. Use the wrong key and you will get nothing.

Listen to **the paladin only**, she knows which key is required.

Use an event handler for the `"hear"` event and determine who said something, and what they said, using the `event` parameters:

- `event.speaker` - who spoke
- `event.message` - what was said

Default Code

```
// Listen to the paladin and fetch the right key.

function onHear(event) {
    // The pet can find the paladin and keys.
    var paladin = pet.findNearestByType("paladin");
    var goldKey = pet.findNearestByType("gold-key");
    var silverKey = pet.findNearestByType("silver-key");
    var bronzeKey = pet.findNearestByType("bronze-key");
    // If event.speaker is the paladin:

        // If event.message is "Gold":

            // The pet should fetch the gold key.

        // If event.message is "Silver":

            // The pet should fetch the silver key.

        // If event.message is "Bronze":

            // The pet should fetch the bronze key.

}

pet.on("hear", onHear);
```

Overview

Pets can use the `findNearestByType(type)` method to find items, friends and enemies.

Flying pets don't need to worry about mines!

Also, pets can pick up more than potions! They can pick up keys!

Your pet can recognize who spoke -- `event.speaker` and what was said -- `event.messege`. `speaker` is a unit, `message` is a string.

Dangerous Key Solution

```
// Listen to the paladin and fetch the right key.

function onHear(event) {
    // The pet can find the paladin and keys.
    var paladin = pet.findNearestByType("paladin");
    var goldKey = pet.findNearestByType("gold-key");
    var silverKey = pet.findNearestByType("silver-key");
    var bronzeKey = pet.findNearestByType("bronze-key");
    // If event.speaker is the paladin:
    if (event.speaker == paladin) {
        // If event.message is "Gold":
        if (event.message == "Gold") {
            // The pet should fetch the gold key.
            pet.fetch(goldKey);
        }
        // If event.message is "Silver":
        else if (event.message == "Silver") {
            // The pet should fetch the silver key.
            pet.fetch(silverKey);
        }
        // If event.message is "Bronze":
        else if (event.message == "Bronze") {
            // The pet should fetch the bronze key.
            pet.fetch(bronzeKey);
        }
    }
}

pet.on("hear", onHear);
```

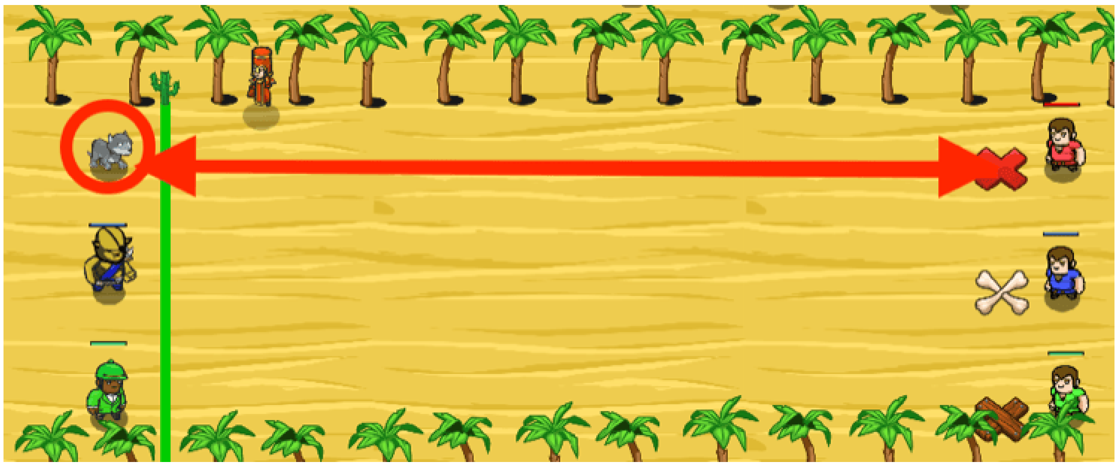
#63. Olympic Race

Level Overview and Solutions

Intro

Your pet has entered a race!

The pet should **ONLY** run when the **referee** says `"Start"` . Do not listen to the other units!



Default Code

```
// The pet must win the race.
// Runners should touch their teams mark and run back.

function onHear(event){
    var referee = pet.findNearestByType("wizard");
    // If the referee is speaker and the message is "Start":

        // Make the pet run to the red mark.

        // Then run back.

}

// Assign the onHear function to handle "hear" events.
```

Overview

Your pet is a participant in a race. Wait for the referee (a wizard) to say the command `"Start"` , then run to the red mark and return, crossing the finish line. **Avoid false starts**, run only after the command. Don't listen to the fans, only the wizard.

To get the information about who was speaking and what was said use the event handler parameter `event` and its properties: `event.speaker` and `event.message` .

If you have the problem with the current level try returning to previous levels and refresh your skills.

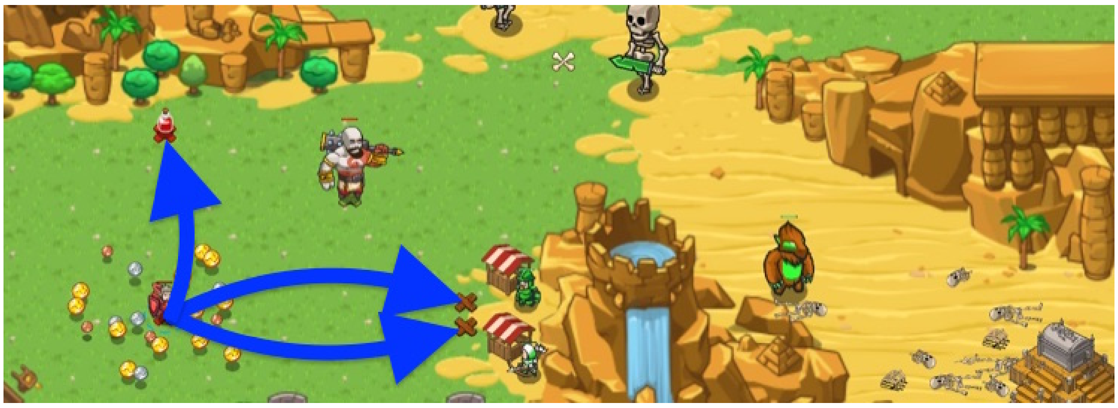
Olympic Race Solution

```
// The pet must win the race.
// Runners should touch their teams mark and run back.

function onHear(event){
    var referee = pet.findNearestByType("wizard");
    // If the referee is speaker and the message is "Start":
    if (event.speaker == referee && event.message == "Start") {
        // Make the pet run to the red mark.
        pet.moveXY(53, 27);
        // Then run back.
        pet.moveXY(6, 27);
    }
}

// Assign the onHear function to handle "hear" events.
pet.on("hear", onHear);
```

#64. Cross Bones

Level Overview and Solutions

Intro



Cross Bones is a battle between the two team's forces.

1. Collect Coins
2. Move near the tents to summon troops
3. Pick up the health potion to heal your guardian
4. Use flags to monitor special eventst that may arise
5. Win!

Default Code

```
// Welcome to Cross Bones!
// Accumulate coins to raise a suitable army.
// Move to the X Markers to hire troops.
// Soldiers cost 20 gold to summon.
// Archers cost 25 gold to summon.
// If your hero has over a certain amount of gold, do something!
// Pick up the potion to heal yourself and the guardian.
// Watch for flags that may appear signaling certain events.

while(true) {
    if(hero.gold >= 45) {
        // If your hero has over a certain amount of gold, do something!

    } else {
        var item = hero.findNearestItem();
        // Gather coins to summon units!

    }
}
```

Overview

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**.

Cross Bones Solution

```
// Welcome to Cross Bones!
// Accumulate coins to raise a suitable army.
// Move to the X Markers to hire troops.
// Soldiers cost 20 gold to summon.
// Archers cost 25 gold to summon.
// If your hero has over a certain amount of gold, do something!
// Pick up the potion to heal yourself and the guardian.
// Watch for flags that may appear signaling certain events.

while(true) {
    if(hero.gold >= 90) {
        // If your hero has over a certain amount of gold, do something!
        hero.moveXY(58, 20);
        hero.moveXY(58, 16);
        hero.moveXY(58, 20);
        hero.moveXY(58, 16);
    } else {
        var item = hero.findNearestItem();
        if(item && item.type != "potion") {
            hero.moveXY(item.pos.x, item.pos.y);
        }
    }
}
```