



# Dragon Blast

An Intermediate Coding Activity for Grades 6 and Up

## Teacher Guide

# Table of Contents

<b>Quick Start</b>	<b>3</b>
Dragon Blast Facts	3
<b>Activity Guide</b>	<b>5</b>
Learning Objectives	5
How to Complete the Puzzles	7
Block Guide	8
Standards Alignment	10
Optional Warm-up Activity: Life Conditionals	12
Puzzles 1-5	13
Puzzles 6-7	16
Puzzles 8-11	18
Puzzles 12-14	21
Puzzles 15-18	23
Puzzles 19-21	26
Puzzles 22-24	27
Puzzles 25-27	30
<b>Activity Wrap-Up</b>	<b>32</b>
<b>Classroom Setup</b>	<b>33</b>
Tracking Student Progress	33
Student Certificates	33
<b>Other Hour of Code Activities</b>	<b>34</b>
<b>Going Beyond an Hour</b>	<b>35</b>
Do More with Tynker	35
Learning Pathways	35
Tynker for Schools	35
<b>About Tynker</b>	<b>36</b>
<b>About the Hour of Code</b>	<b>36</b>

# Quick Start

## Dragon Blast Facts

- Web Address: [tynker.com/hour-of-code/dragon-blast](https://tynker.com/hour-of-code/dragon-blast)
- Coding skill level: **Intermediate**
- Recommended grade level: **Grade 6+**
- Time Required: **60 Minutes**
- Number of modules: **27**
- Coding language: **Block-based, Python, JavaScript**
- Localization: **English, Simplified Chinese, Spanish, German, French, Japanese**

## Welcome!

Welcome to the Hour of Code 2017! If you're new to Tynker, check out our [Hour of Code](#) page to see all Tynker activities. New this year are Dragon Blast and Space Quest, the first Tynker Hour of Code activities available in Spanish, Chinese, German, French, and Japanese as well as English.

## What is Dragon Blast?

Dragon Blast is a puzzle-based activity where students complete advanced coding puzzles and learn computational thinking skills along the way. They play as a dragon hatchling, and go on a treasure hunt in a magical forest.

Dragon Blast is a simplified version of a curriculum created by Tynker for the Everyone Can Code program by Apple. For more information about using the full course in your classroom, check out the iBook [Get Started with Code 2](#) from Apple.

## Who is this activity for?

Dragon Blast is intended for students in grades 6 and up with some coding experience. Students who are younger and have less coding experience should check out Space Quest.

## What will my students learn?

The coding concepts covered in this activity are Algorithms, Debugging, Loops, Decomposition, Abstraction, Functions, and Conditionals. The puzzle sequence maps to CSTA Computer Science standards 1B-A-5-4, 1B-A-3-7, 1B-A-6-8, and 2-A-5-6. For a complete list of standards, see the [Standards Alignment](#) section of this guide.

## What devices do I need?

Each student needs to have a desktop computer, laptop computer, or Chromebook with an internet connection and an up-to-date browser. No downloads are required. If not enough devices are available, students can work in pairs on the same device.

## How should I prepare to teach Dragon Blast?

The best way to prepare for an Hour of Code is to know the activity. Take half an hour to try Dragon Blast on your own beforehand. If you need help, feel free to consult the [Activity Guide](#), which contains a full answer key and block glossary.

## How can Tynker help me manage my Hour of Code?

Tynker has several free features for registered teachers that will help you manage your Hour of Code. If you set your students up with a Tynker classroom, you will be able to track their progress through Dragon Blast and print certificates for them to keep. For more information, see the [Classroom Setup](#) section of this guide.

## How do I change my language settings?

Tynker will automatically detect your language preferences through your browser. If you are using a language that is supported by Dragon Blast, Tynker will use that language. A language selection feature will be implemented soon.



# Activity Guide

This activity is designed for self-directed learning. The puzzles are as engaging as they are challenging. Aside from the optional Warm-up Activity, you will not have to lead your class through Dragon Blast. Your goal will be to help students out individually.

Tynker puzzles increase in complexity, and students may need help occasionally. The best way to help students proceed is to give them hints or clues that allow them to arrive at the answers themselves. If you give correct answers directly, your students won't learn as much as they otherwise could. If you need any help answering student questions, see the [General Information](#) section, which explains the code blocks and the Puzzle interface.

Before your Hour of Code, if you have extra time, get your students in the coding mindset with a [Warm-up Activity](#). This is a discussion about conditionals in everyday life.

Here is a sample schedule:

- 15 Min: Warm up with Life Conditionals
- 45 Min: Complete Dragon Blast
- If you have more time, choose from over 30 Tynker activities based on interest, grade and experience.

## Learning Objectives

Your students will be using logical skills and computational thinking to manipulate code.

Dragon Blast is organized into sections of 2-5 puzzles, corresponding to the following concepts:

**Algorithms:** An algorithm is a step-by-step set of instructions to solve a problem. As they make algorithms, your students will learn how to tailor their solutions to the problem at hand.

**Debugging:** A bug is an error in a piece of code and debugging is the process of finding and fixing that error. As they spot bugs, your students will learn to critically examine code.

**Loops:** A loop is a set of code that gets repeated a certain number of times or until a certain condition is met. As they work with loops, your students will learn to analyze patterns.

**Decomposition:** Decomposition is breaking down complex problems into smaller problems to make them easier to solve. As they decompose problems, your students will learn to handle more difficult puzzles.

**Abstraction:** Abstraction is finding the commonalities within a set so that it is easier to think about that set. As they find shared qualities, your students will improve their understanding of patterns.

**Functions:** A function is a group of code commands that has been given a name. As they use function blocks, your students will learn to combine steps into singular processes.

**Conditionals:** A condition is something that can be checked to see if it is true or false. As they use conditional statements, your students will learn to consider multiple scenarios.

Click the links above to jump to the Answer Keys for each section.

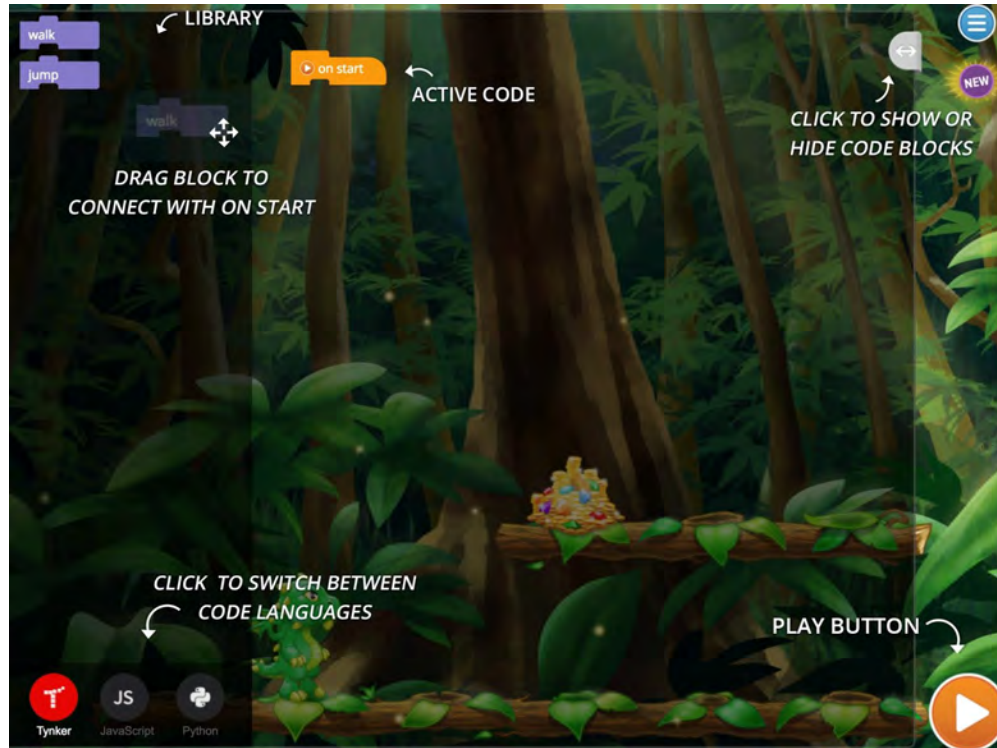
# How to Complete the Puzzles

## Getting Started

To get started, have your students open a browser tab to this URL:

[tynker.com/hour-of-code/dragon-blast](https://tynker.com/hour-of-code/dragon-blast)

## The Workspace



In each puzzle, an overlay will appear containing the workspace. The darker section on the left is the library and shows which types of blocks are available for use for this particular puzzle. In this situation, the only blocks available are the “walk” and “jump” blocks. The lighter section on the right is the active code area that your students will place their working code in. In this situation, there is already a “on start” code block in the active code area. To add a code block to the active code, simply drag the block from the library and connect it to a block in the active code area.

## Visual Code Blocks, JavaScript, and Python

A code block is a block representing a piece of code that your students can use to make their program. The text in the code block has three modes that can be selected at the bottom left of the workspace: **Tynker visual blocks**, **JavaScript**, and **Python**. For younger and inexperienced students, we recommend using Tynker blocks but students who want a challenge may want to



use the other languages. Note that our answer key uses the Tynker blocks, but if you need a reference for JavaScript and Python, we have included the corresponding code in the Blocks Guide.

## Running Your Code

To run the code, your students should click on the play button on the bottom right of the screen. This will remove the workspace and the dragon should follow the code blocks that your students added to the active code area. While the dragon is moving, the code will be shown at the upper left. Your students can watch the code blocks execute by following either the green outlines. The green will move as each code block is executed.

## Common Issues


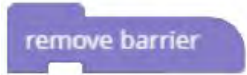
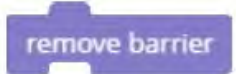



- Disconnected blocks
  - Make sure your students have connected all of their blocks and that the blocks are connected to the “on start” block. Code that is not connected to the “on start” block will not be executed. Functions definitions blocks do not need to be connected to the “on start” block.
- Using too few or too many blocks
  - Drag more blocks over if the cadet does not reach the end. If your students accidentally add too many blocks or a block that they do not want, then they can drag the block from the right to the left and a trash can symbol will show up. Once they release the block, the block will be removed from their code.
- Deleting the “on start” block
  - If students accidentally delete a block that is not available in the library, they need to restart the puzzle by clicking the button on the top right with the three lines and then selecting the refresh button to reset their code.
- Incorrect sequencing
  - For all of the puzzles, there are particular actions that have to happen at particular times. For some of the puzzles there is only one solution for the puzzle. For others there may be multiple ways to organize the code blocks to get the solution, especially when loops are involved. If your students are struggling, suggest that they read through the code blocks one by one and trace what will happen to their cadet with their finger.





# Block Guide

Block	Description
<p>On Start</p>  <p><b>JavaScript:</b> <code>function _on_start() { }</code>  <b>Python:</b> <code>def _on_start():</code></p>	<p>This is an event block that will run all code attached to it the play button is pressed. Students must attach their code to the bottom of the “on start” block for it to run.</p>
<p>Walk</p>  <p><b>JavaScript:</b> <code>walk();</code>  <b>Python:</b> <code>walk()</code></p>	<p>Moves the dragon one space forward.</p>
<p>Jump</p>  <p><b>JavaScript:</b> <code>jump();</code>  <b>Python:</b> <code>jump()</code></p>	<p>Makes the dragon jump straight up and move one space forward to land on a new platform.</p>
<p>Eat</p>  <p><b>JavaScript:</b> <code>eat();</code>  <b>Python:</b> <code>eat()</code></p>	<p>Makes the dragon eat the bug that is directly in front of it.</p>
<p>Blast</p>  <p><b>JavaScript:</b> <code>blast();</code>  <b>Python:</b> <code>blast()</code></p>	<p>Makes the dragon breathe fire or ice to destroy an object in front of it.</p>
<p>Turn around</p>  <p><b>JavaScript:</b> <code>turn_around();</code>  <b>Python:</b> <code>turn_around()</code></p>	<p>Makes the dragon face opposite direction.</p>

<p>Repeat</p>  <p><b>JavaScript:</b> for (var i = 0; i &lt; 10; i++){ }</p> <p><b>Python:</b> for i in range(0, 10):</p>	<p>Everything inside of this block is repeated a certain number of times. The amount of times that the code repeats is the number that is next to the word repeat (the count). To change the number of repeats, click on the count and enter the number that you want. This is called a counting loop because it counts the number of times that the loop has been repeated and stops when it hits the inputted number.</p>
<p>Function Definition</p>  <p><b>JavaScript:</b> function remove_barrier(){ }</p> <p><b>Python:</b> def remove_barrier():</p>	<p>This block is similar to an “on start” block in that it does not connect to another block on the top. It is it’s own set of code. All the code attached to the bottom of the “function definition” block will run each time the corresponding “function command” block is used.</p>
<p>Function Command</p>  <p><b>JavaScript:</b> remove_barrier();</p> <p><b>Python:</b> remove_barrier()</p>	<p>Executes the code stored in the “function definition” block for the corresponding function and must be attached to another block. The “function definition” block and “function command” block must have the same name.</p>
<p>If</p>  <p><b>JavaScript:</b> if ( _Icefly() ){ }</p> <p><b>Python:</b> if ( _Icefly() ):</p>	<p>If the statement in the between the “if” and the “then” is true, then the code inside the block will execute. Otherwise, the code inside the block will be ignored. This block will perform the check every time it is executed.</p>
<p>While</p>  <p><b>JavaScript:</b> while( _Treasure() ){ }</p> <p><b>Python:</b> while( _Treasure() ):</p>	<p>Note that in this course, the “while” blocks say “while” instead of “repeat while”. Everything inside of this block is repeated until the statement represented by the “false” is no longer true. If the statement never changes from true to false, the code will be repeated forever. If the statement is never true, the code will not be run. This is called a conditional loop because it relies on a condition being false to stop.</p>
<p>Not</p> 	<p>This is a conditional block that negates the statement that replaces the “false” that is</p>

**JavaScript:** !\_Treasure()

**Python:** not \_Treasure()

currently there. This block can be added to “if” and “while” blocks. In the case of a “while” block, the loop will run until the statement inside “false” is false and will stop when it becomes true.

## Standards Alignment

Dragon Blast is mapped to the following standards:

### CSTA Computer Science:

- 1B-A-5-4, 1B-A-3-7, 1B-A-6-8, 2-A-5-6

### Common Core CCSS-Math:

- MP.1, 5.G.1, 5.G.2, 6.NS.6, 1.OA.1, 2.OA.1, 1.OA.2, 2.OA.2, 1.OA.3, 2.OA.3, 1.MD.4

### Common Core CCSS-ELA:

- 1.RF.1, 2.RF.1, 1.RF.4, 2.RF.4, 5.RF.4, 6-8.RST.3, 6-8.RST.4, 6-8.RST.7, 1.RI.3, 2.RI.3, 1.RI.6, 2.RI.6, 1.RI.7, 2.RI.7, 1.RI.10, 2.RI.10, 3.RI.3, 4.RI.3, 3.RI.5, 3.RI.7, 4.RI.7, 3.RF.3, 4.RF.3, 3.RF.4, 4.RF.4, 1.L.3, 2.L.3, 2.L.6, 3.L.1, 4.L.1, 3.L.2, 4.L.2, 3.L.3, 4.L.3, 3.L.4, 4.L.4, 3.W.3, 4.W.3, 3.W.4, 4.W.4, 3.W.6, 4.W.6

# Optional Warm-up Activity: Life Conditionals

If you have more than 50 minutes available, try this discussion as an unplugged coding activity.

Make a diagram on the board with two columns.

- Label the left column “Events”. This will be a list of events that demand some response.
- Label the right column “Responses”. This will be a list of appropriate reactions for each event.
- Discuss some simple examples with your class. For each event, write down the appropriate response in the right column. In the middle of these two columns, draw arrows connecting event to response.
  - Ask your class what they are supposed to do in the event of an earthquake.
  - Ask about what to do when the fire alarm goes off.
  - Ask about what to do if there is a power outage.
- Now discuss some more familiar examples:
  - What do you do if you have a homework assignment due tomorrow?
  - What do you do when your shoe is untied?
  - What do you do when your hands are dirty?
  - What do you do if you bump into someone accidentally?
- Continue connecting events and responses with arrows.

Explain that these situations all have something in common: they are all conditional statements.

- Write “If” above the left column and “Then” above the right column. Demonstrate that everything on the board makes sense when you read it as “If (A), then (B).”

Say “We understand this as part of normal life. If you notice something, you react to it. But computers need to be programmed to act this way. We program computers to notice things and react to them using conditional statements.”

- If you are holding Shift, the letters you type become capital letters.
- If you are watching a YouTube video and you press ‘K’, the video will pause-- but if you are typing a search into YouTube’s search bar, the ‘K’ key functions normally and does not pause the video!
- If you quickly press the Home button on an iPhone or iPad twice, the device does something different than when you only press Home once.

As you transition into playing Dragon Blast, explain to your students that they are going to have to program their dragon to notice and react using conditionals. The first several puzzles won't have any conditionals, but the later ones will.

# Puzzles 1-5

Concepts: Algorithms

## Overview

These puzzles are designed to introduce your students to the Tynker coding environment and to practice with simple coding blocks. Students will also be exposed to sequencing and creating algorithms. A **sequence** is the order in which instructions are performed and an **algorithm** is a step-by-step set of instructions to solve a problem. For each puzzle, your students will need to come up with an algorithm containing a specific sequence of instructions for the dragon to follow to reach the pile of treasure.

## Puzzle 1: Select a Dragon

Your students must choose their character. They have the choice between 3 dragons: Jasper, Grooper, and Harper. They will need to click the egg many times to make it hatch.





## Puzzle 2: Jump

Move the dragon to the pile of treasure using “walk” and “jump” blocks.



## Puzzle 3: Blow Fire

Use the “eat” and “blast” blocks to eat the bug and destroy the wood. Move the dragon to the pile of treasure.





## Puzzle 4: Jump and Blow Fire

Combine the “walk”, “jump”, “eat”, and “blast” blocks to move the dragon to the pile of treasure.



## Puzzle 5: Change Direction

Use the “turn around” block and the old blocks to move the dragon to the pile of treasure.



# Puzzles 6-7

Concepts: Algorithms, Debugging

## Overview

These puzzles are designed to introduce your students to debugging. A **bug** is an error in a piece of code and **debugging** is the process of finding and fixing that error. In each puzzle, your students will be given a puzzle and a piece of code that has a bug in it. They will need to fix the bug so that the dragon can reach the end of the level. There are a couple of ways your students can approach debugging:

- Run the buggy code to see what happens before modifying the code.
- Read through the buggy code and use a finger to trace what should be happening to the dragon as each code block is executed. They may also want to say the steps outloud to understand what is happening.
- Ignore the buggy code and write out by hand what they would do if they were approaching the puzzle as they did with previous puzzles. Then compare that code to the buggy code.

Debugging is a major part of coding and helps to reinforce understanding of the concepts that they learned in previous puzzles.

## Puzzle 6: Ice Breath

Debug the code so that the dragon can get past the fire to the pile of treasure.

Debugging Steps: Add a "walk" block after "eat". Add a walk after "blast". Add a "walk" block after "jump".





## Puzzle 7: Jump.

### Grow. Jump.

Debug the code so that the dragon can get to the pile of treasure.

Debugging steps: Move the bottom “jump” block to after the “eat” block. Add a second “walk” block to the end.



# Puzzles 8-11

Concepts: Algorithms, Debugging, Loops

## Overview

These puzzles are designed to introduce your students to loops. A **loop** is a set of code that gets repeated a certain number of times or until a certain condition is met. Loops allow programmers to use reduce the amount of code that they have to write. These puzzles will use a particular kind of loop called a **for loop** which is a loop that repeats a section of code a certain number of times. To complete the puzzles, your students will need to look for a repeating pattern that can be used to reach the pile of treasure. That repeating pattern will be what they place inside the loop block.

### Puzzle 8: Repeat

Use the “repeat” block to move the dragon to the pile of treasure.



## Puzzle 9: Triple Jump

Use the “repeat” and “jump” blocks to move the dragon to the pile of treasure.



## Puzzle 10: Blow. Blow. Blow.

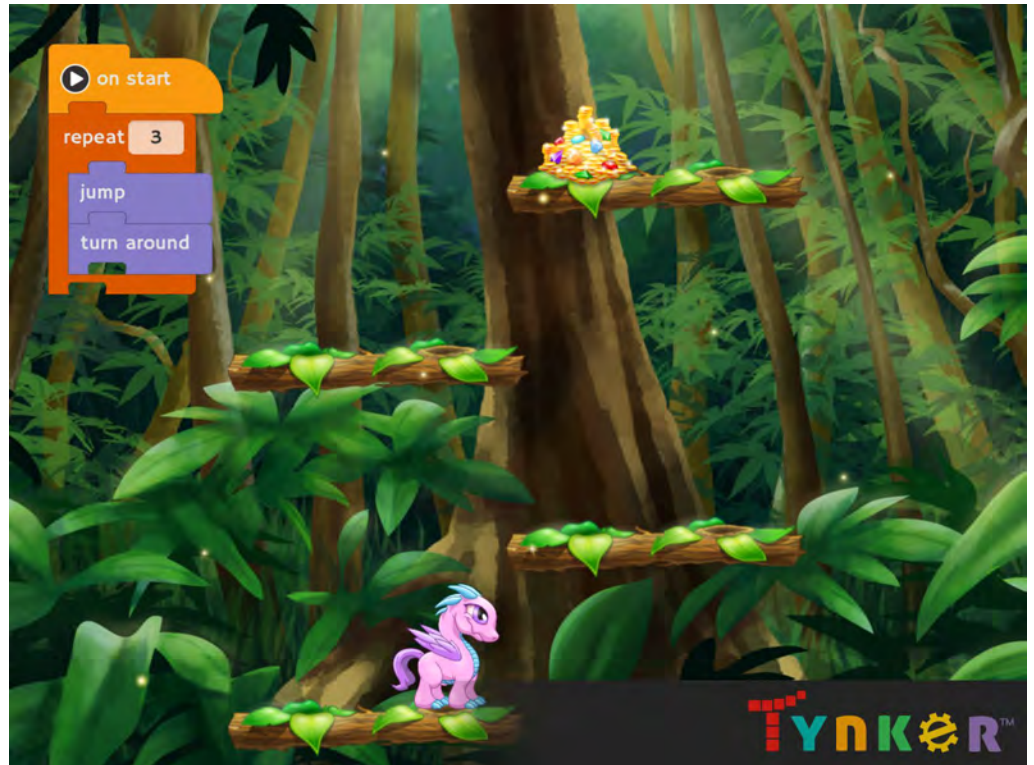
Use the “repeat”, “eat”, and “blast” blocks to move the dragon to the pile of treasure.





## Puzzle 11: Jump. Turn Around. Repeat.

Use the “repeat”, “jump”, and “turn around” blocks to move the dragon to the pile of treasure.



# Puzzles 12-14

Concepts: Algorithms, Debugging, Decomposition

## Overview

These puzzles are designed to introduce your students to the concept of decomposition.

**Decomposition** is breaking down the complex problems into smaller problems to make completing the problem easier. These puzzles are more challenging than before and your students will have to use all of the skills that they have learned in previous puzzles to reach the end. There will likely be a lot of testing and rearranging their code. If your students are struggling, ask them to break down the problem and suggest they solve one small problem such as getting to the end of one platform. Once they have a solution to all of the smaller problems, they can put it all together to complete the level.

### Puzzle 12: Jump Up. Jump Down.

Use all the blocks to move the dragon to the pile of treasure.





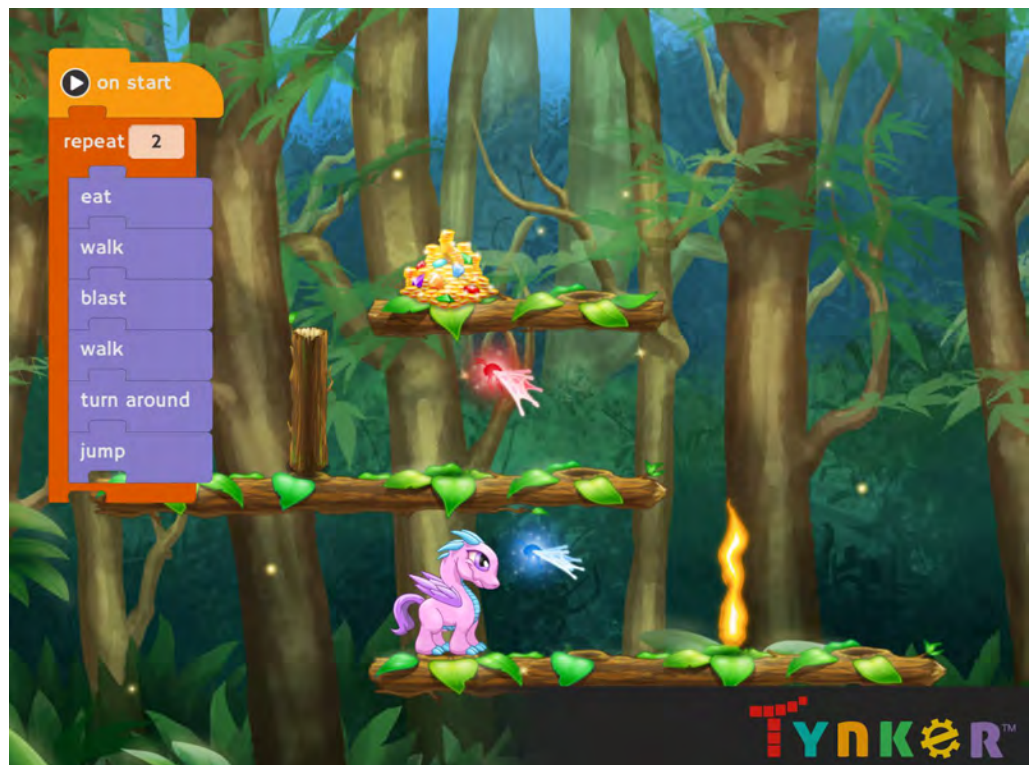
### Puzzle 13: Ice Breath. Grow.

Use all the blocks to eat the different bugs and move the dragon to the pile of treasure.



### Puzzle 14: Eat. Blast. Turn Around. Jump.

Use all the blocks to eat the bugs, navigate the platforms, and move the dragon to the pile of treasure.



# Puzzles 15-18

Concepts: Algorithms, Debugging, Loops, Decomposition, Abstraction

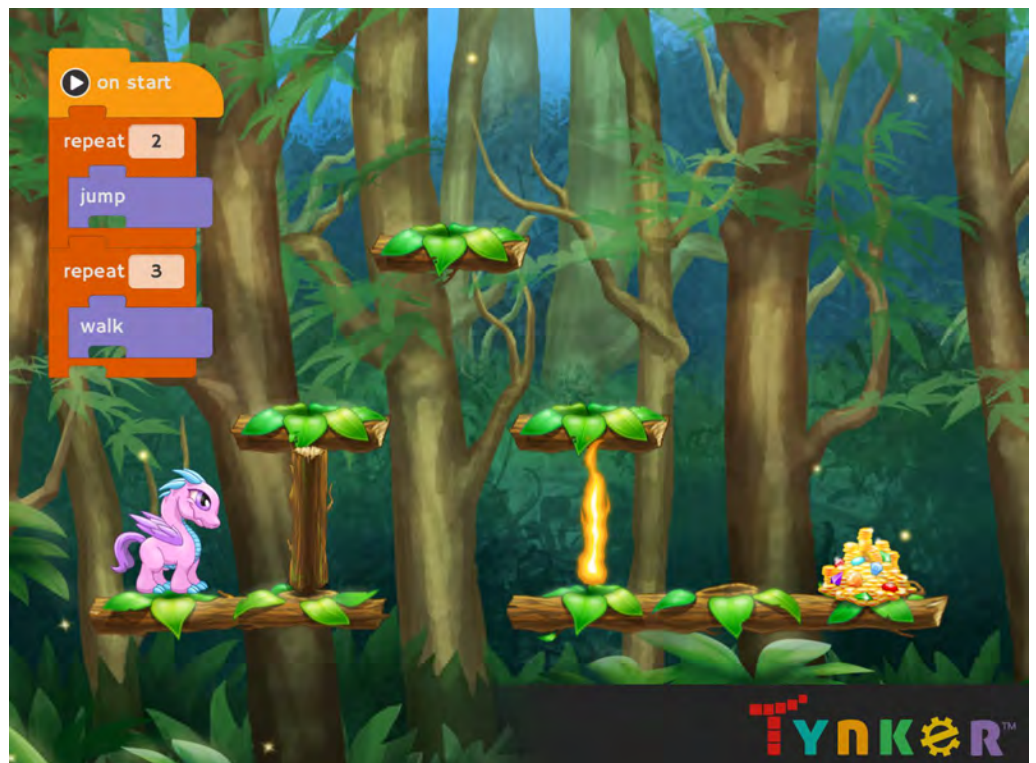
## Overview

These puzzles are designed to introduce your students to the concept of abstraction.

**Abstraction** is finding the commonalities within a group so that it is easier to think about that group. In coding, abstraction is useful when the same actions need to be performed on different items. If we can group the items by similarities, then we can have code that works on every item in a group. Your students will first practice using blocks that they have used before in a slightly different way. Then, they will get to practice abstraction using a story template. The template already has each of the items divided into groups such as title and location. Your students will be able to change the input for each of these items and create a whole new story using the same template. They will be able to see how using abstraction to group items makes it much easier to switch out code.

### Puzzle 15: Jump Up. Jump Down.

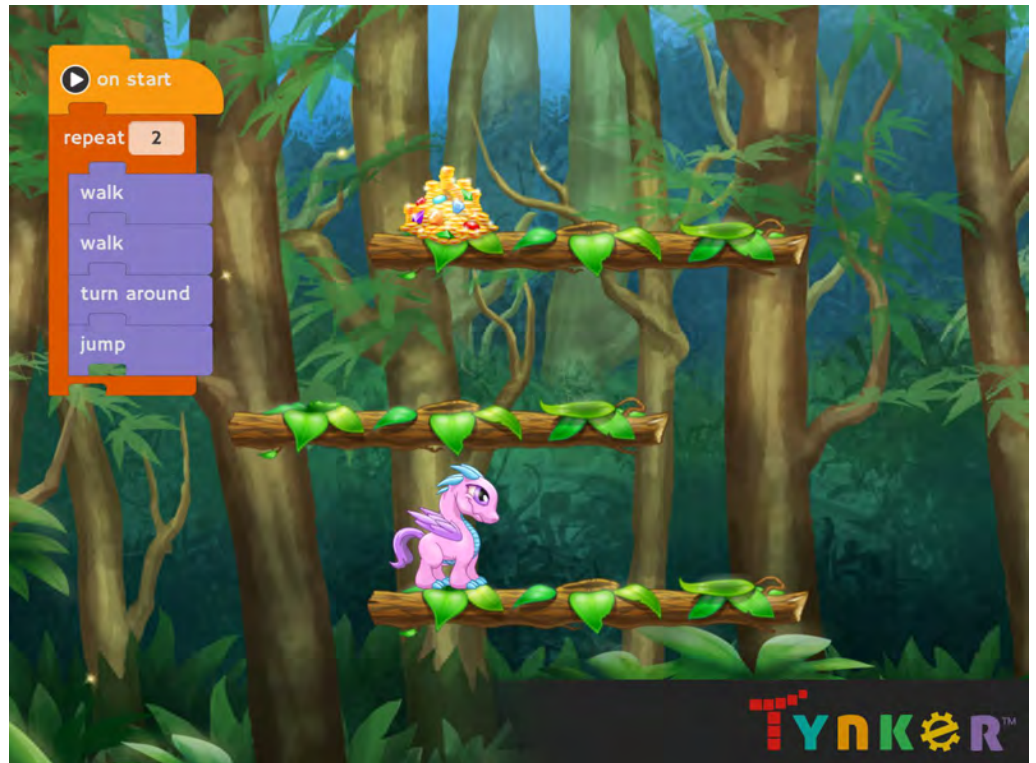
Use all the knowledge you have gained to move the dragon up and down the platforms to reach the pile of treasure.





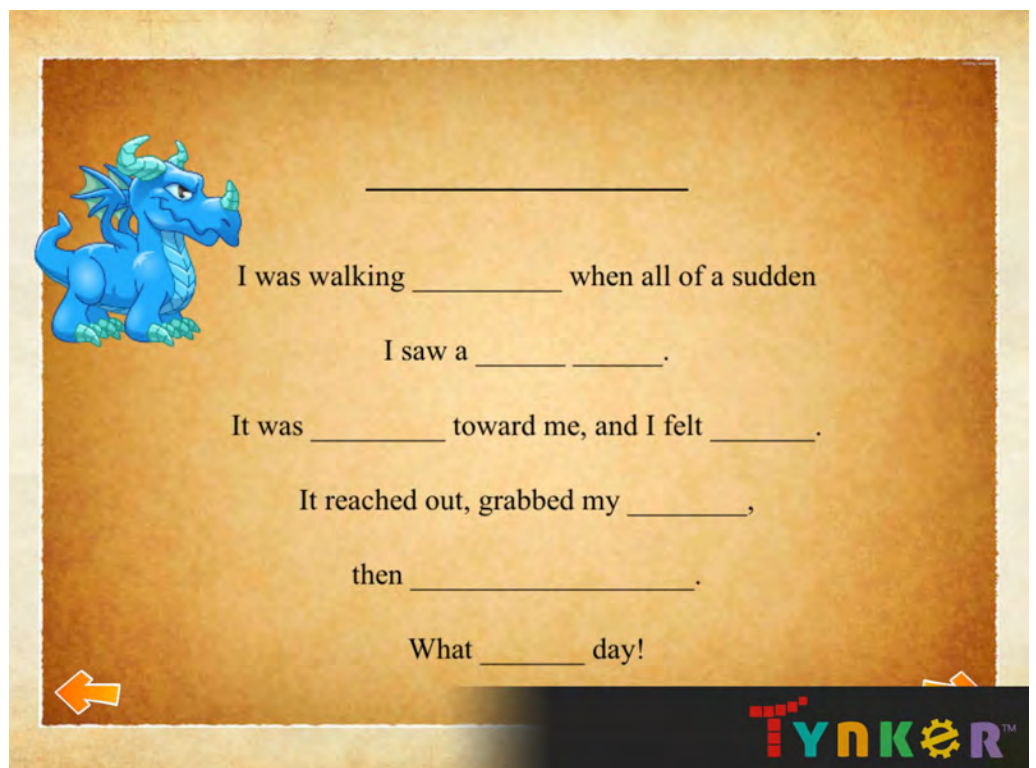
## Puzzle 16: Walk. Turn Around. Jump.

Find the pattern and  
move the dragon to the  
pile of treasure.



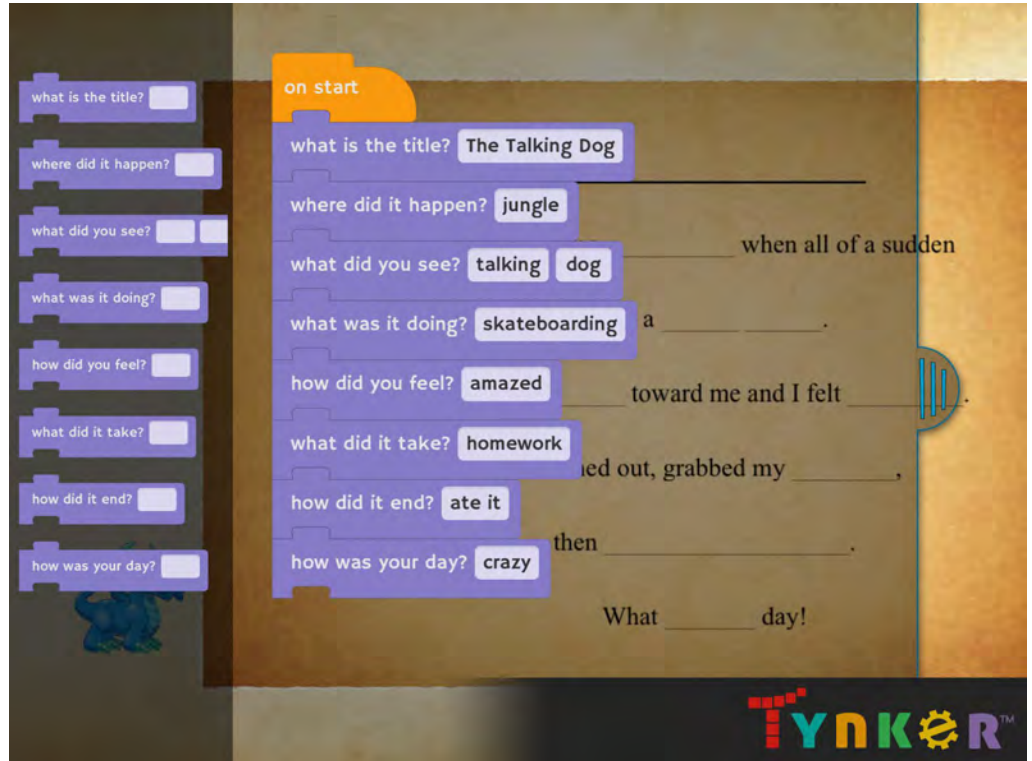
## Puzzle 17: A Dragon's Tale Introduction

Observe how abstraction  
is used in coding by  
looking at the story  
templates.



## Puzzle 18: A Dragon's Tale

Use the templates to make various stories. Once the students are done experimenting, they can move on to the next puzzle. This is an open ended project and students can spend as much time on this puzzle as they would like.



The screenshot shows the Tynker interface for creating a story. On the left, a sidebar lists prompts for a story template. The main area displays a story template with a code editor on the right.

**Story Template Prompts:**

- what is the title?
- where did it happen?
- what did you see?
- what was it doing?
- how did you feel?
- what did it take?
- how did it end?
- how was your day?

**Code Editor Content:**

```

on start
  what is the title? The Talking Dog
  where did it happen? jungle
  what did you see? talking dog
  what was it doing? skateboarding
  how did you feel? amazed
  what did it take? homework
  how did it end? ate it
  how was your day? crazy
  
```

**Story Template Text:**

\_\_\_\_\_ when all of a sudden  
 a \_\_\_\_\_  
 toward me and I felt \_\_\_\_\_  
 ed out, grabbed my \_\_\_\_\_  
 then \_\_\_\_\_  
 What \_\_\_\_\_ day!

# Puzzles 19-21

Concepts: Algorithms, Debugging, Loops, Decomposition, Abstraction, Functions

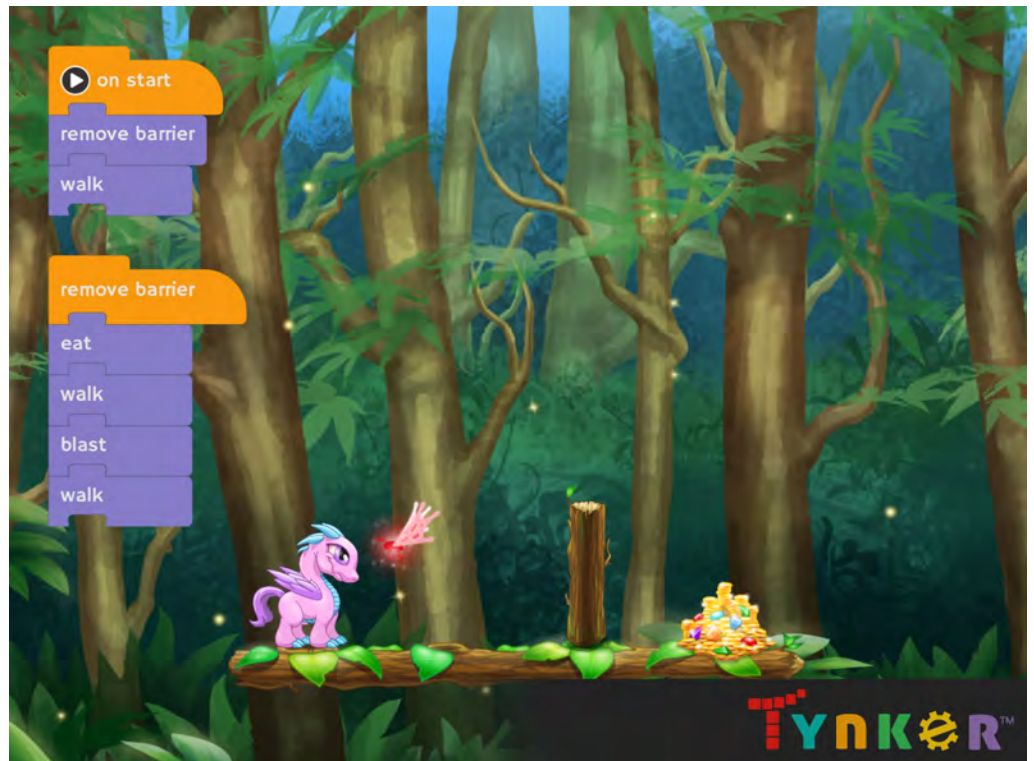
## Overview

These puzzles are designed to introduce your students to functions. A **function** is a group of code commands that has been given a name. To use the set of commands in a function, your students will use the name of the function to execute it. Functions are useful when you want to perform the same actions many times but you do not want to have a lot of repeated code blocks. Similar to loops, functions make your code more efficient. The function will be defined in a “function definition” block and to use the function, your students will need to use the “function command” block. These blocks will be given to your students so they will not have to name the functions themselves.

### Puzzle 19:

#### Remove Barrier

Write a function “remove barrier” that will handle removing the piece of wood in front of the dragon. Use the function to move the dragon to the pile of treasure.





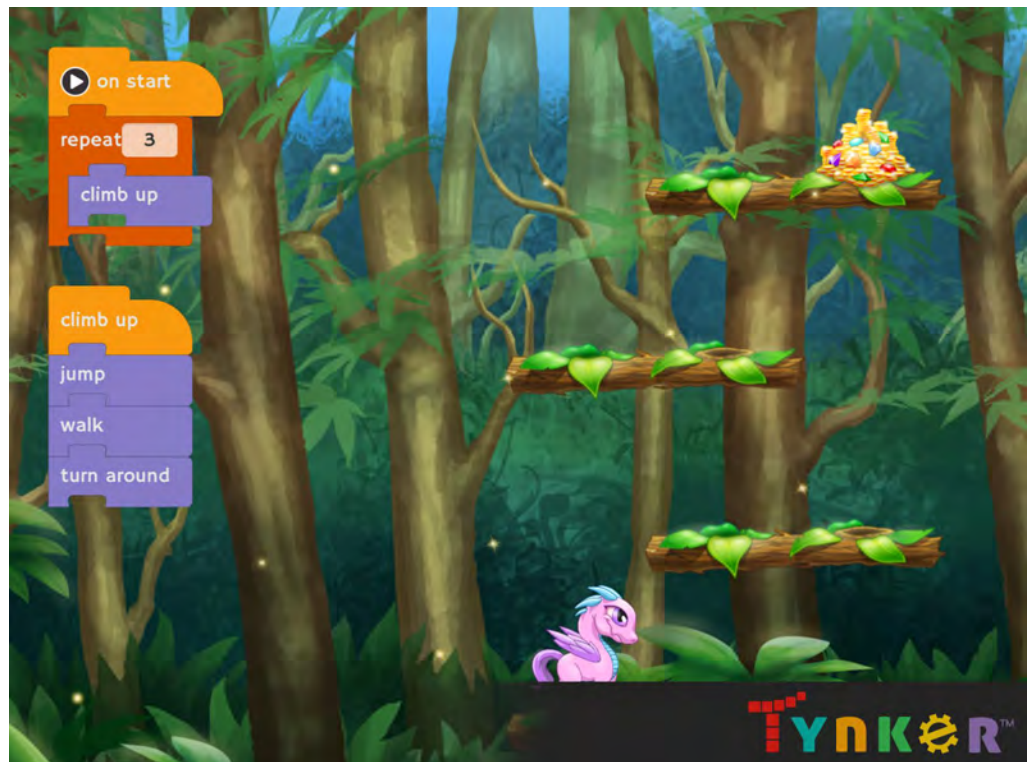
## Puzzle 20: Remove Barrier Twice

Use the “remove barrier” function to destroy the barriers and move the dragon to the pile of treasure.



## Puzzle 21: Detect Pattern and Climb Up

Write a function “climb up” that will handle making the dragon jump up the platforms. Use the “climb up” command block to execute the function as many times as necessary.



# Puzzles 22-24

Concepts: Algorithms, Debugging, Loops, Decomposition, Abstraction, Functions, Conditionals

## Overview

These puzzles are designed to introduce your students to conditions and conditional statements. A **condition** is something we can check if it is true or false. For example, we can check if the dragon has reached the pile of treasure. If the dragon is at the treasure, the condition is true, if not the condition is false. A **conditional statement** gives certain instructions when certain conditions are true. For these puzzles, your students will be using if statements. An **if statement** is a conditional statement that gives instruction only if the condition is true. For example, if you said the if statement “If you’re happy, clap your hands”, your students would only clap their hands if they are happy. Otherwise, the command “clap your hands” would be ignored. There are also if-else statements. **If-else statements** are the same as if statements but they have the extra “else” component which contains the instructions that are to be followed if the condition is not true.

### Puzzle 22:

#### Moving Icefly

Use the “if” block to check if there is a bug in front of the dragon for it to eat. Move the dragon past the fire and get to the pile of treasure.

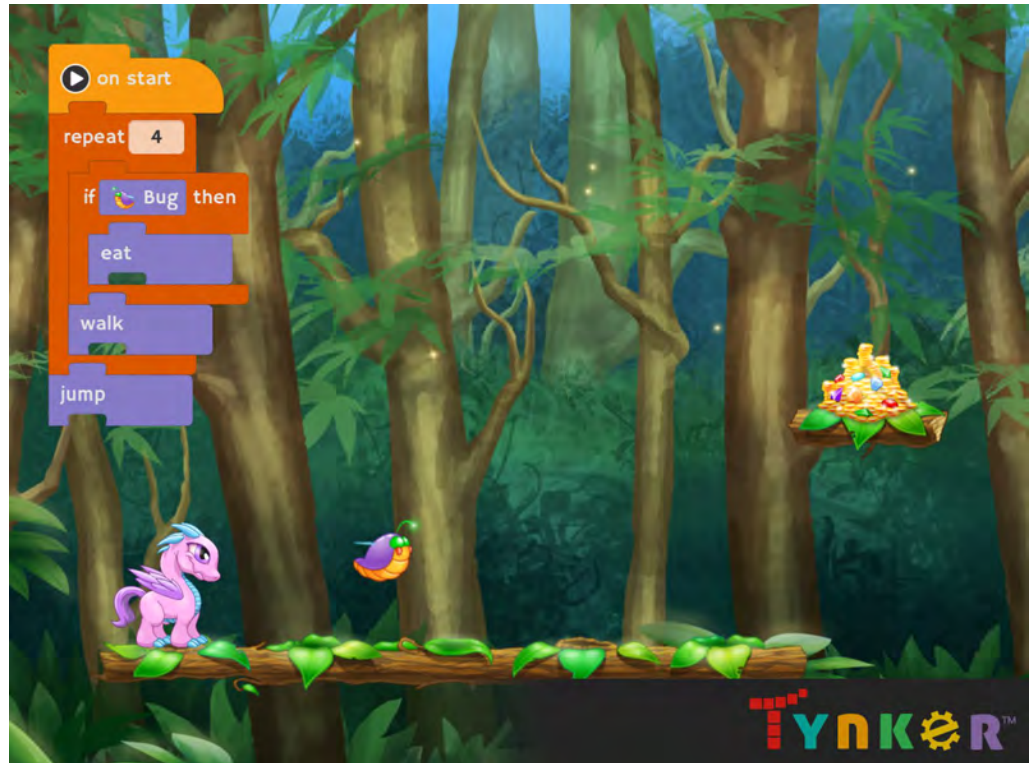




## Puzzle 23:

### Moving Bug

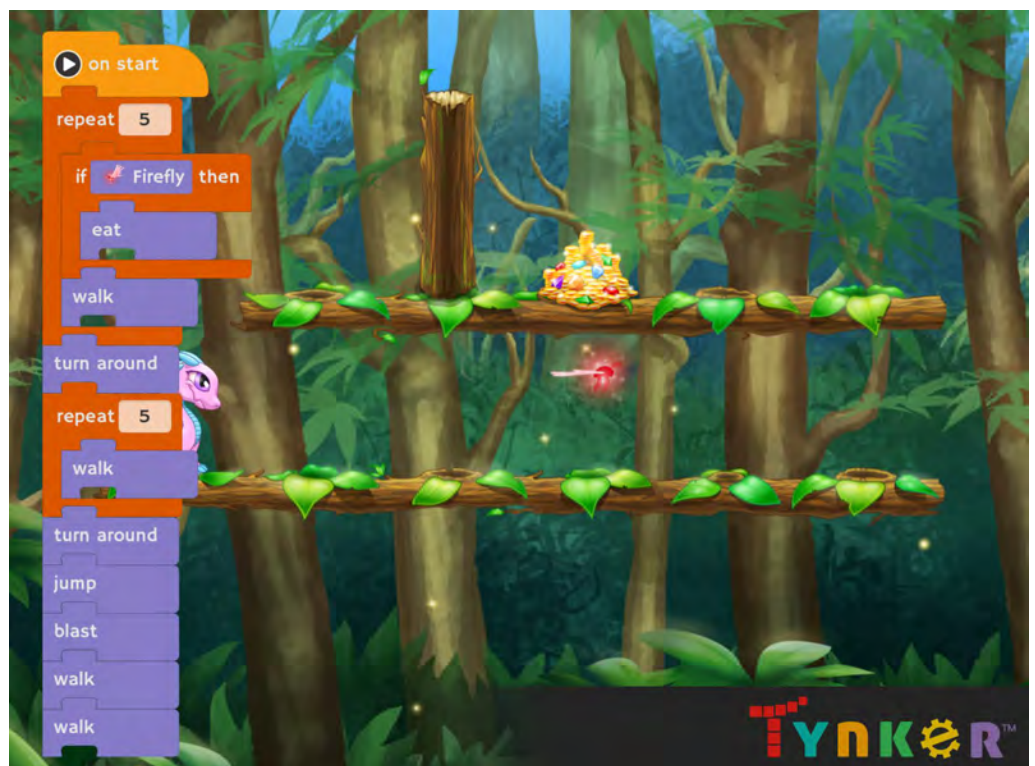
Use the “if” block to check if there is a bug in front of the dragon for it to eat. Move the dragon to the next platform and get to the pile of treasure.



## Puzzle 24:

### Moving Firefly with Jump

Use the “if” block to check if there is a bug in front of the dragon for it to eat. Move the dragon to the pile of treasure.



# Puzzles 25-27

Concepts: Algorithms, Debugging, Loops, Decomposition, Abstraction, Functions, Conditionals, Conditional Loops

## Overview

These puzzles are designed to introduce your students to using conditional expressions to control loops. A **while loop** is a loop that will run while a specific condition is true. Each time the loop reaches the bottom of the code in the loop, it will go back to the beginning and check if the condition is true or false. If the condition is true, the loop will run again. If the condition is false, the loop will stop executing and the program will move onto the code located after the loop. A negated condition can be used in place of a regular condition. This means that the loop will execute while the condition is false and will stop when the condition is true.

### Puzzle 25: Jump While Treasure Not Found

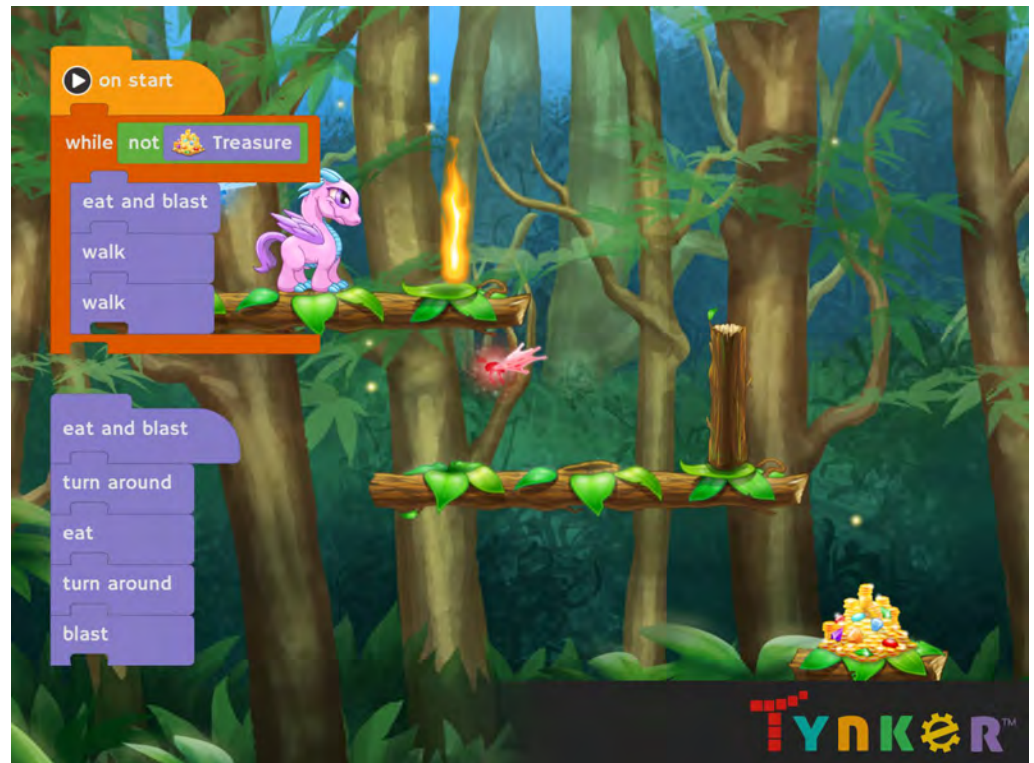
Use a while loop to move the dragon to the pile of treasure.





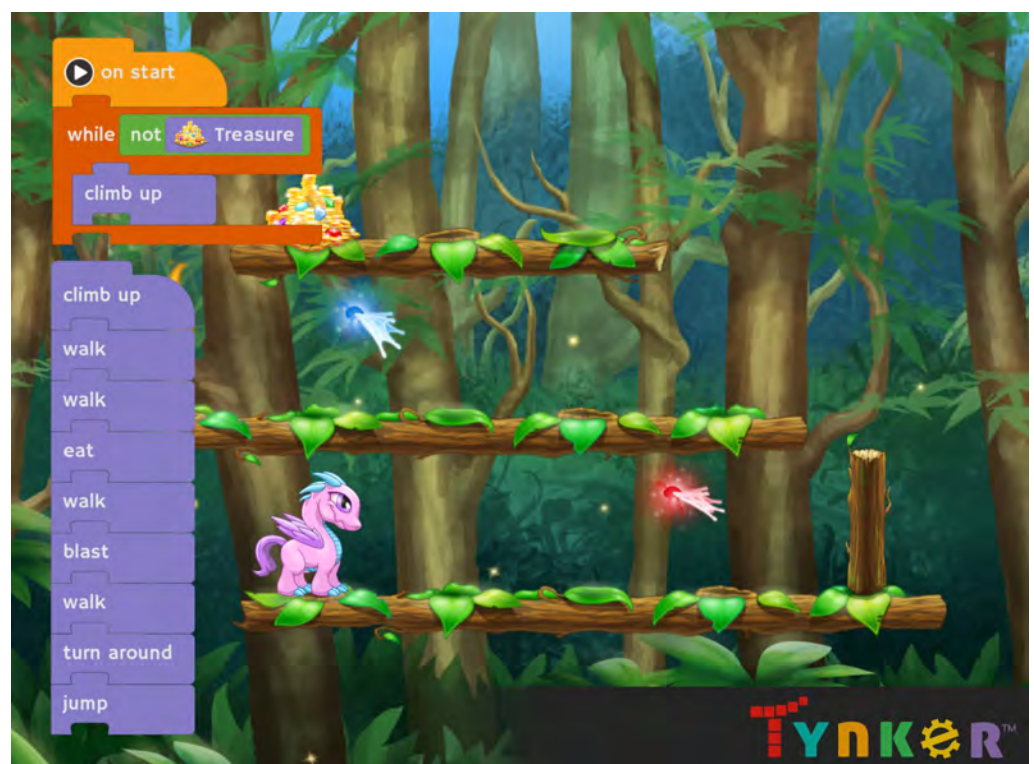
## Puzzle 26: Eat and Blast

Write a function called “eat and blast” and use that function to move the dragon to the pile of treasure.



## Puzzle 27: Climb Up

Write a function called “climb up” and use that function to move the dragon to the pile of treasure.



# Activity Wrap-Up

Once your students finish the final puzzle, they will have completed their hour of code! Over the course of the 27 puzzles, your students will have explored the importance of developing algorithms, analyzed and tested code through debugging, improved their code by finding patterns and using loops, practiced breaking complex problems down into smaller problems through decomposition, created templates through abstraction, organized their code into functions, and learned about how computers use conditionals to make decisions. These skills are fundamental to programming and your students will now have a good programming foundation with which to build on.

Other than their importance in programming, the concepts covered in this Hour of Code activity can be carried over into other subjects such as math, science, music, and literature. Scientists must perform the steps in an experiment in a particular sequence. Composers condense the length of sheet music by creating loops with phrases that indicate to the performer to repeat a specific part of music. Mathematicians correct their work by analyzing what their calculations should do and correcting errors just as programmers debug their code. Authors can create a large number of stories from a small set of conditional statements by writing adventure stories where different events happen depending on the choices the reader makes. Check out Tynker's [Hour of Code STEM Activities](#) page to get started using computer science throughout your curriculum.

We hope that your students enjoyed their programming with Tynker and will continue to develop their coding skills outside of Hour of Code. Read on for more information about how to continue your students' coding journey.

# Classroom Setup

[Click here](#) to access Tynker's Quick Start guide for teachers. It only takes a few minutes to make a free Teacher account and a Tynker classroom for your students. If you are already set up with Google Classroom or Clever, you can use those services to automatically sync student accounts and classroom information with Tynker.

If you set your students up with a Tynker classroom, you will be able to:

- Track your students' progress through Dragon Blast
- Print certificates of completion for your students to keep
- Save students' progress to their accounts, so that they can continue coding at home
- View [teacher guides and answer keys](#) for all Tynker Hour of Code activities
- Access a [free introductory coding course](#) for your class
- Give your students access to all of Tynker's free content

## Tracking Student Progress

Once you setup your students with a Tynker classroom, you'll be able to observe their progress in Dragon Blast using your Teacher Dashboard.

- Go to your Teacher Dashboard and select your classroom.
- Navigate to the "Gradebook" tab, then choose "Hour of Code."

### Gradebook

[Hour of Code](#) [Mobile Puzzles](#) [Everyone Can Code](#) [Lesson Progress](#) [Quiz Results](#) [Concepts Mastery](#)

- You will be able to see the amount of puzzles your students have completed for each Hour of Code activity, including Dragon Blast. Refresh the page to update it.

## Student Certificates

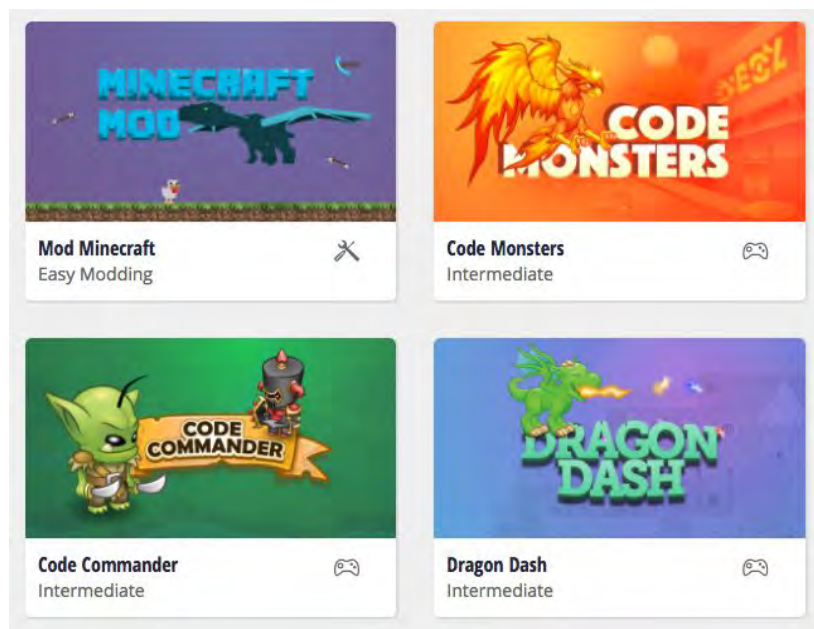
Every time a student completes an Hour of Code coding activity in Tynker, they earn a badge that is added to their certificate. There are 29 badges that they can earn, so they'll be really excited to complete all the activities!

While signed in to a Teacher account, you can print certificates by clicking on a classroom from your Teacher dashboard, clicking the "Gradebook" tab, going to "Hour of Code", and clicking the "Print All Certificates" button. This will only print certificates for student accounts assigned to the selected classroom.



# Other Hour of Code Activities

Tynker offers many other tutorials for the Hour of Code, including intermediate-level adventures [Code Monsters](#) and [Code Commander](#). These game-like tutorials are your best options if you are interested in having another Hour of Code with the same group of students. For teachers who are interested in the creative applications of coding, we recommend [Brick Breaker](#) and [Solar System](#). Check out the main [Tynker Hour of Code](#) page to see all the tutorials!





# Going Beyond an Hour

## Do More with Tynker

If your students enjoyed an Hour of Code, they're sure to enjoy the rest of what Tynker has to offer! Tynker offers a complete premium solution for schools to teach Computer Science. Over 300 hours of lessons are available to take K-8 students from block coding to advanced text coding. We offer tons of resources for teachers, including comprehensive guides, free webinars, and a forum to connect with other educators.

## Learning Pathways

With Tynker, kids don't just acquire programming skills-- they can explore the world of possibilities that coding opens up. Tynker has several interest-driven learning paths that make coding fun, both inside and outside the classroom.

- **Coding and Game Design:** The Tynker app allows your students to access the Tynker Workshop, a powerful tool for crafting original programs. They can even share their work with other kids in the Tynker Community.
- **Drones and Robotics:** Tynker integrates with connected toys, including Parrot drones and Lego WeDo robotics kits, so kids can see their code come to life.
- **Minecraft:** Tynker integrates with Minecraft so your students can learn coding through a game they love. Tynker offers skin and texture editing, as well as a custom Mod Workshop that lets kids try their original code in Minecraft.

## Tynker for Schools

Used in over 60,000 schools, our award-winning platform has flexible plans to meet your classroom, school, or district needs. All solutions include:

- Grade-specific courses that teach visual coding, JavaScript, Python, robotics and drones
- A library of NGSS and Common Core compliant STEM courses that are great for project-based learning
- Automatic assessment and mastery charts for the school, class and student level
- Easy classroom management with Google Classroom and Clever integration
- Professional training, free webinars and other teacher training resources

# About Tynker

Tynker is a creative platform designed to make coding fun to learn and easy to teach. Tynker's mission is to empower kids to become makers and equip them with computing skills for today's digital world. Over 60 million kids have begun their coding journey with Tynker!

## About the Hour of Code

The Hour of Code is a global learning event in which schools and other organizations set aside an hour to teach coding. The event is held every December during Computer Science Education Week. You can also organize an Hour of Code year-round.

The goal of the Hour of Code is to expand access to computer science education for people of all backgrounds. Learning computer science helps students develop logic and creativity, and prepares them for the changing demands of the 21st century.

Tynker has been a leading provider of lessons for the Hour of Code since the event began in 2013. Since then, over 100 million students from 180 countries have finished an Hour of Code. For more information, visit the [Hour of Code website](#).

**Happy coding!**