Last updated: 08/31/2018

TEACHER GUIDE

Arena Levels

Level: Beginner, Intermediate, Advanced

Time: 1 or 2 x 50 to 60-minute sessions

Overview

The Arena levels, when used as course-end capstones, provide opportunities for students to creatively apply all the concepts they have learned to develop a program that solves a problem. Students first develop an algorithm that will defeat a computer opponent, then they compete against classmates, refining their algorithm to beat the level as quickly as possible. The friendly competition in this level is intended to motivate students to apply an iterative development process, creating and testing multiple possible solutions. Students can also work collaboratively using a pair programming approach, which may make the competition more comfortable for some students.

There are four Arenas in CodeCombat:

Arena	Course	Concepts
Wakka Maul	Introduction to Computer Science	Sequencing, syntax, strings, while True loops, variables
Power Peak	Computer Science 2	Functions, if statements, events, parameters

Arena	Course	Concepts
Cross Bones	Computer Science 3	Boolean logic, nested conditionals, arithmetic
Summation Summit	Computer Science 4	Arrays, while loops, for loops, objects

Materials

- Engineering Cycle Worksheet (http://files.codecombat.com/docs/resources/EngineeringCycleWorksheet.pdf)
- Optional: Pair Programming Guide (https://codecombat.com/teachers/resources/pair-programming)
- Optional: Python Syntax Guide (http://files.codecombat.com/docs/resources/Course1PythonSyntaxGuide.pdf) or JavaScript Syntax Guide (http://files.codecombat.com/docs/resources/Course1JavaScriptSyntaxGuide.pdf)

Learning Objectives

- Use an iterative process to develop a program that solves a problem.
- Test and debug a program.

Standards

- **CSTA: 1B-AP-13** Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences.
- **CSTA: 1B-AP-15** Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended.
- **CSTA: 3B-AP-09** Implement an artificial intelligence algorithm to play a game against a human opponent or solve a problem.

- **CCSS-Math: MP.1** Make sense of problems and persevere in solving them.

- **CCSS-Math: MP.2** Reason abstractly and quantitatively.
- **CCSS-Math: MP.6** Attend to precision.

Opening Discussion (5 minutes): *Introduction to Arenas*

Explain

Tell students that they're going to put all their learning together today in a special activity called an Arena. Explain and demonstrate how the Arena works, making sure to cover the following points:

- They'll be writing an artificial intelligence program to beat a complicated level in a race against an opponent. They'll test and revise their program over and over the get the best time they can. Encourage them to submit code, observe the output, and look for places where revisions could help a goal to be achieved more quickly, help a player to stay alive longer, etc. Then they should make the changes and submit again, as many times as they like.
- First, they should click to select the Red (Human) or Blue (Ogre) team. (We suggest randomly assigning half of the students to each team.)
- The first time they play, they should choose "Warm-Up" to play against the computer. They should keep revising and improving their program until it is good enough to beat the computer.
- Once they beat the computer, they can choose "Easy" to play against their classmates.

Review the Engineering Cycle

Remind students that engineering is all about solving problems, and the first rule of engineering is that no one gets it right the first time. That’s where the Engineering Cycle comes in:

DECOMPOSE: Understand and break apart the problem. What is the goal of the level? What smaller goals do you see along the way?
PLAN: Choose one part of the problem to solve first. What do you need the computer to do? Plan a solution in plain English or pseudocode. Use a flowchart or storyboard to stay organized.
IMPLEMENT: Write the solution to each part of your problem in code. Tip: If you've learned about functions, define a different function to solve each part!
TEST: Run your code! Does it solve the problem the way you intended? If not, redesign. Does it work without errors? If not, trace through it to find and fix the bug(s), then test again. Once it works, move on to the planning and implementing the next part!

Provide each student with a copy of the Engineering Cycle Worksheet (http://files.codecombat.com/docs/resources/EngineeringCycleWorksheet.pdf) that they can use to plan their program once they navigate to the level.

Discuss

Use one or more of the following discussion questions to help prepare students for success:

What steps will you follow to plan and create your program?

Sample Responses:

> I'll use the Engineering Cycle. I'll decompose the problem by finding the big goal and breaking it into smaller subgoals. I'll choose one subgoal to start with and plan out an algorithm to solve it. Then I'll write my algorithm in code, and start testing and debugging until it works the way I want it to.

> Observe the level, locate the goal, identify items to collect and enemies to defeat, plan the shortest route to complete those tasks, then use the coding skills I've learned to write a program step-by-step that will complete all the tasks and get to the goal.

What will you do if your program doesn't beat the computer the first time?

Sample Response:

> If it doesn't work, I'll rerun it and watch to see where it goes wrong, then I'll try to find a way to improve that part and resubmit the code. If it still doesn't work, I'll try it again! It's the Engineering Cycle.

Coding Time (40-50 mins)

Have students navigate to the arena level. They should take a few minutes to complete the Engineering Cycle Worksheet, then complete the level at their own pace. Circulate and assist as they work.

Good to Know

- Some students may be uncomfortable with competition, especially given that the rankings are visible to the class. Consider using Pair Programming - competition is often more comfortable when you have a partner.
- Students will only compete against the AI and other students in the same CodeCombat class (not strangers).
- Once students have beaten one of the AIs, they will be put into the class rankings.
- Red teams only fight against blue teams, and there will be top rankings for each.
- Once students have submitted code, other students can click the “Fight” link next to any student in the ranking to challenge that student!

- If you leave your teacher account on the arena ladder page, it will simulate more matches between your students.

Look Out For:

- The Arena is more open-ended than the regular levels. If students are unsure of how to get started, remind them that programming is an iterative process and guide them toward decomposing the problem into simpler pieces, planning a solution for just one part of the level at a time.
- If a student is frustrated at losing, encourage them to analyze the winning player's strategy. What can they learn from it, and how can they use it to improve the next iteration of their own code?

Closure (5 minutes)

Use one or more of the following questions to prompt reflection on the lesson. You can facilitate a short discussion, or have students submit written responses on Exit Tickets.

In CodeCombat, you have to plan all your hero's actions in advance, then let the hero carry them out all at once. This is different from most video games, where you directly control the hero and make decisions as you go. How do you feel about the difference? For example, which is more fun? Which is harder? How does your strategy change? How do you handle mistakes?

Sample Responses:

> CodeCombat is harder because I have to think so many steps ahead!
> It's a fun kind of hard!

> In this game, I get to look through the whole level first and plan out how I want to beat it. Then I get to design a way to make my plan work. It feels different than making it up as I go along in regular video games.

What did you do when your code didn't beat your opponent? How did you decide what changes to make?

Sample Responses:

> I reran the code and watched to see if I could take any shortcuts. Then I changed the code and ran it again to see if it helped.

> I looked for ways to stay alive longer. I called more friends to help and picked up more potion. Adding those things to my program helped me make it to the end.