Code.org curricula (Blockly-based)

Andrea Sterbini – sterbini@di.uniroma1.it

CC BY NC SA

Code.org

Built with Blockly: a JavaScript library for visual languages

Code.org (and AppInventor.mit.edu)

Fine-grained activities within a CONSTRAINED environment for a C.S. curriculum

(initially less freedom … later full environment)

Initial language

NO local variables

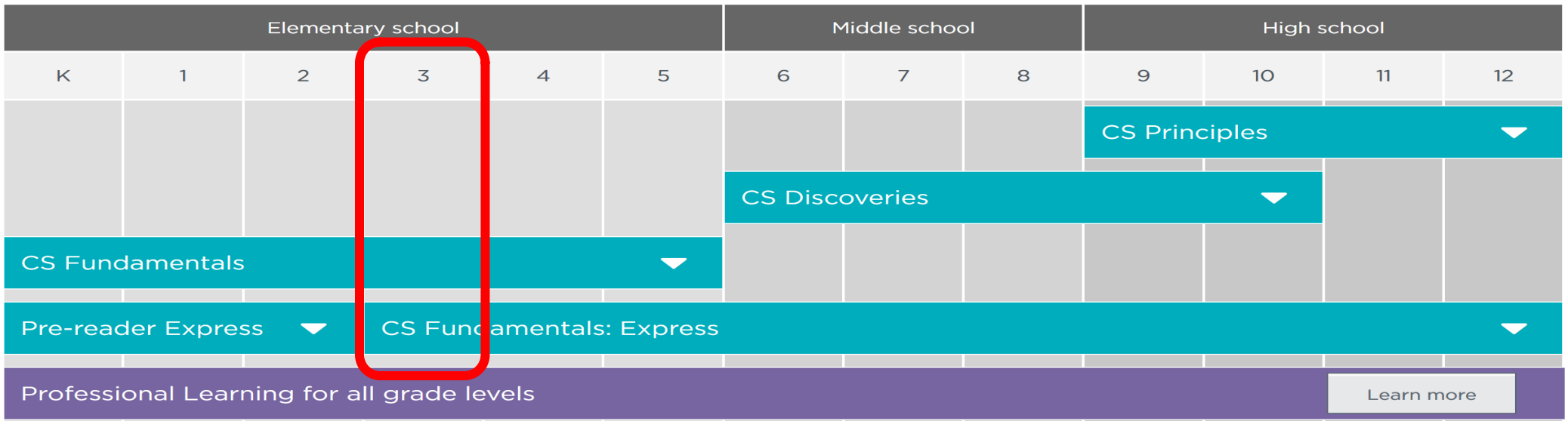
NO personal agent attributes

Procedures (NO return value)

Possibility of static data type enforcement with visual clues

Puzzle-like connectors with different shapes and colors: Actors, numbers, text, booleans

Complete curriculum from Elementary to High school (USA)



A course tailored to students of each year:

E.g. Course D for 3rd grade (K3): algorithms, nested loops, while loops, conditionals, and events.
Beyond coding, students learn about digital citizenship.

Both "unplugged" and programming activities are used

Example: Course D for 3rd grade (K3 = 8-9 y old)

Topic: DIGITAL CITIZENSHIP

Lesson 1: Password Power-up ***(Common Sense Edu. | Unplugged)***

Stronger, more secure online passwords are a good idea for everyone. But how can we help kids create better passwords and actually remember them? Use the tips in this lesson to help kids make passwords that are both secure and memorable.

Topic: SEQUENCING (algorithms)

Lesson 2: Graph Paper Programming **(Unplugged)**

In this context-setting lesson, students use symbols to instruct each other to color squares on graph paper. By "programming" one another to draw pictures, students get an opportunity to experience some of the core concepts of programming in a fun and accessible way.

Lesson 3: Introduction to Online Puzzles **(Sequencing | Debugging | Loops | Angry Bird | Collector | Artist | Harvester)**

In this skill-building lesson, students will practice their sequencing and debugging skills in maze puzzles.

… continue SEQUENCING

Lesson 4: Relay Programming (Unplugged | Relay Programming | Algorithms)

This **context-setting** lesson will begin with a short lesson on debugging and persistence, then will quickly move to a race against the clock as students break into teams and work together to write a program one instruction at a time.

Lesson 5: Debugging with Laurel (Debugging | Bug | Collector | Laurel)

In this **skill-building** lesson, students will **practice debugging** in the "collector" environment. Students will get to practice reading and editing code to **fix puzzles with simple algorithms, loops and nested loops**.

Topic: EVENTS

Lesson 6: Events in Bounce (Event | Bounce)

In this **context-setting/skill-building** lesson, students will learn what **events** are and how programmers use them in video games. Students will **build a game that they can customize with different speeds and sounds**.

Lesson 7: Build a Star Wars Game (Events | Star Wars)

In this **skill-building** lesson, students will practice **using events to build a game** that they can share.

Lesson 8: Dance Party (Timed Events | Music)

In this **skill-building** lesson, students will program an **interactive dance party**.

Topic: LOOPS

Lesson 9: Loops in Ice Age (Loops | Scrat | Ice Age)

This context-setting/skill-building lesson will quickly introduce students to loops.

Lesson 10: Drawing Shapes with Loops (Loops | Artist)

This skill-building lesson builds on the understanding of loops from the previous lesson and doubles as a debugging exercise for extra problem-solving practice.

Lesson 11: Nested Loops in Maze (Nested Loops | Loops | Bee | Maze)

In this skill-building lesson, students will learn how to program a loop inside of another loop.

Topic: CONDITIONALS

Lesson 12: Conditionals with Cards ***(Conditionals | Unplugged)***

In this context-setting lesson, students will write conditional (if/else) statements to state the rules of simple card games.

Lesson 13: Looking Ahead with Minecraft **(Conditionals | Minecraft)**

This skill-building lesson gives students the chance to practice concepts that they have learned up to this point and get their first experience with conditionals!

Lesson 14: If/Else with Bee **(Conditionals | Bee)**

In this skill-building lesson, your class will continue to code with conditionals, allowing them to write code that functions differently depending on the specific conditions the program encounters.

...continue CONDITIONALS

Lesson 15: While Loops in Farmer (While Loops | Loops | Farmer)

In this skill-building lesson, students will be working to fill holes and dig dirt in Farmer, but they will not know the size of the holes or the height of the mounds of dirt. To solve these puzzles, students will use a new kind of loop.

Lesson 16: Until Loops in Maze (Until Loop | Maze | Angry Bird | Zombie)

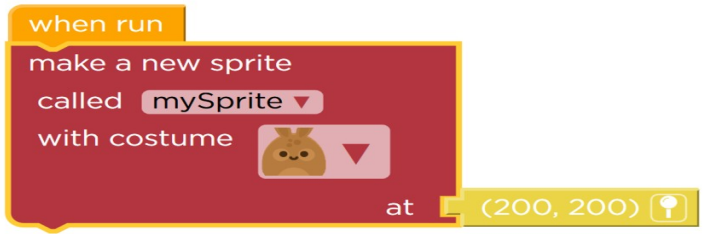
In this skill-building lesson, students will learn about "until" loops. Students will build programs that have the main character repeat actions "until" they reach their desired stopping point.

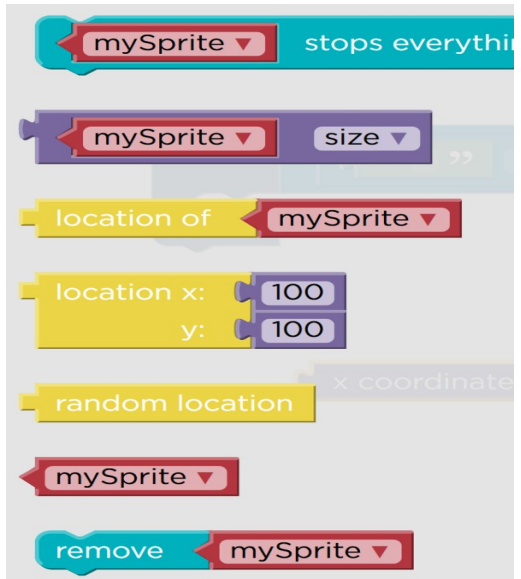
Lesson 17: End of Course Project (Play Lab | Event)

This capstone lesson takes students through the process of designing, developing, and showcasing their own projects!

Visual language
User interaction and common features

Visual choosers to simplify input: Sprite's "costumes", colours, angles, positions, sound/music, …



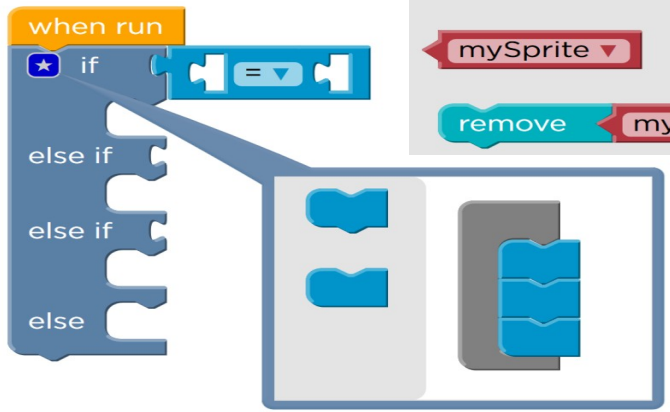


Typed connectors: positions, sprites, numbers, conditions, text

Extensible if (if, elif, elif, …, else)

Counted loops (with counter)

Show corresponding JavaScript code



Made with Blockly

A JavaScript library to build visual languages (initially by Google)

Easy way to define new types of blocks with:

- Typed inputs (int, string, object, list, boolean, …) and outputs
- Conversion of the resulting code to many programming languages (JavaScript by default, but also Lua, Python, PHP, Dart, …)
- You can also define new blocks visually by using Blockly

The resulting JavaScript can be evaluated to interact with the page

- Labyrinths, Harvesting robots, Games, Simulations, …

Used in: code.org, appinventor.mit.edu, programmailfuturo.it, open-roberta.org, and many more …

Environments: Artist: turtle graphics

Single program, no events

Single agent (Pen), NO concurrency/events

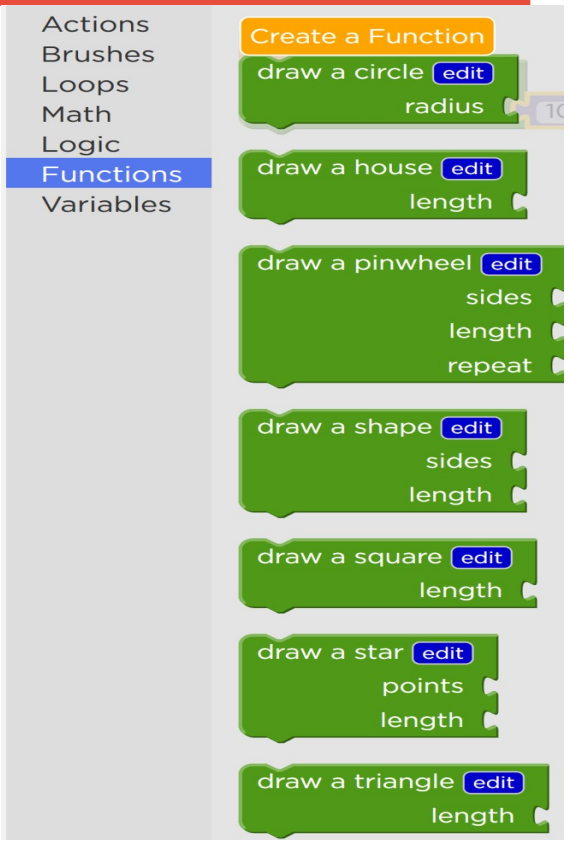
New: PARAMETRIC procedures

Automatic redraw/run when parameters change

With some examples of editable procedures/drawings

RECURSION! (demo)

Useful tricks:
- pen-up => set alpha = 0
- pen-down => set alpha = 255
- or just use “Jump”



Sprite Lab: multiple interacting Actors

Single initial program (e.g. to create Sprites and scene)

(Multiple) actors reacting to simple events (but NO messages)

Concurrent execution of events

Multiple threads for same event (demo)

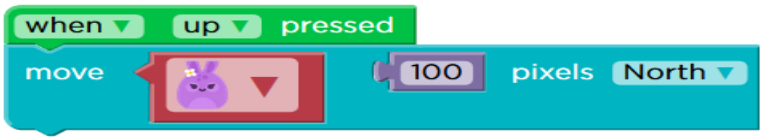
Simple procedures (without parameters!)

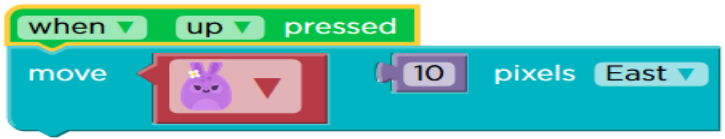
Simple "behaviors" common to all agents

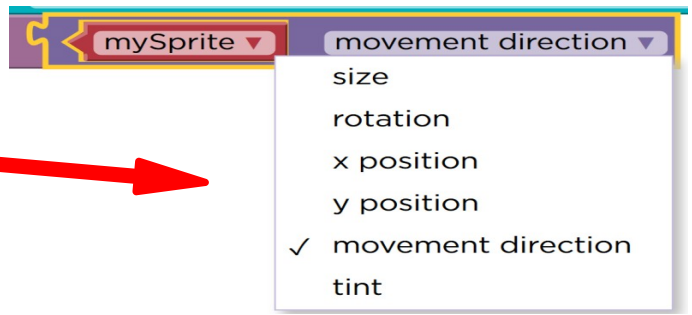
Fixed Sprite properties

Global variables

NO lists

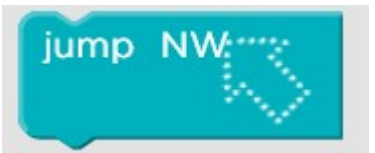






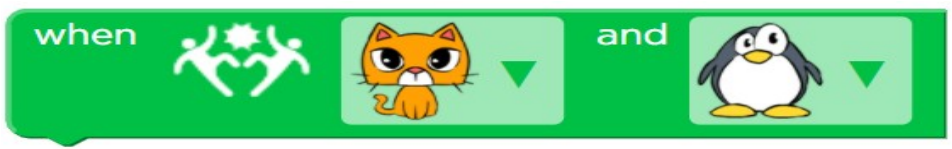
Pre-reader versions!

Artist: single program, NO events, NO variables, NO if-then-else, fixed angles/distance, draw/jump/stickers, fixed loop



Play Lab: behaviors attached to agents (when up/touched/hit)

NO variables, simple commands, NO if-then-else, fixed repetition

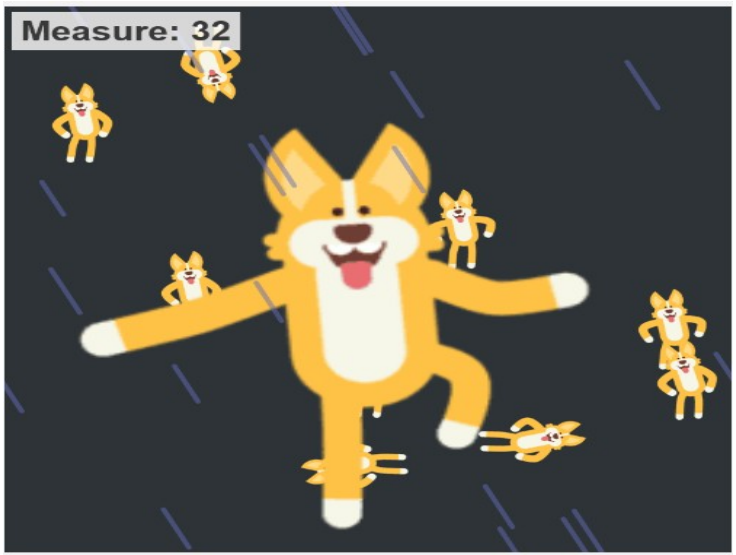


Dance Party: music-sync animation

Animated “dancers” with dance moves (clap, dab, gagnam, …)

Background effects (rain, disco lights, …)

Initial Setup + Events: keyboard / timing / music (demo)



Music-related events/conditions

- if dancer is clapping/if measure>8
- move dancers wrt bass/mid/treble

Dance-related conditions (if doing “clap”)

Concurrency (multiple identical events)

NO messages (demo)

Procedures (NO functions)

Game Lab: build a “game” app

You implement a single function called by the game refresh loop (NO Events!!!)

Animated sprites + Grouped sprites/movement

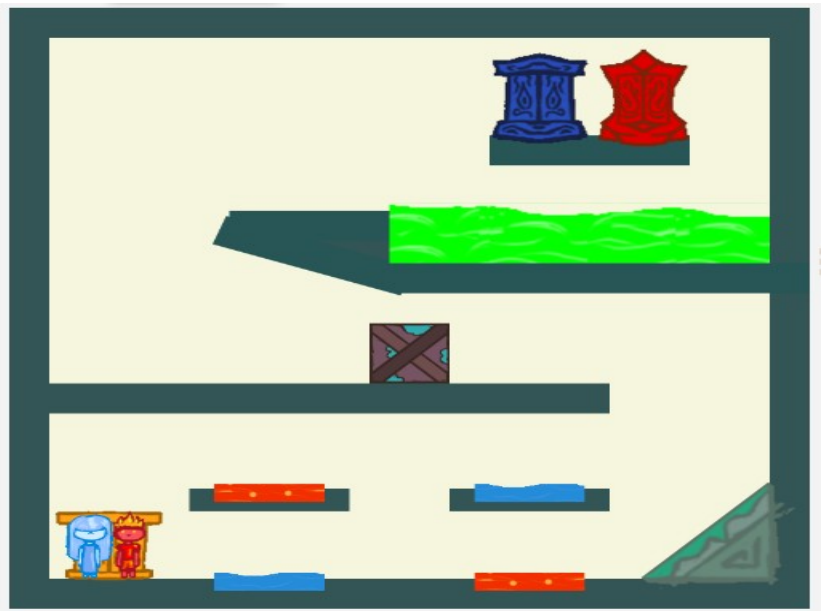
Drawing primitives

Sprite interaction primitives
(collide, displace, bounce …)

Variables as game status
(positions, points, lives)

You must implement ONLY the “paint” function to update the screen

(demo)

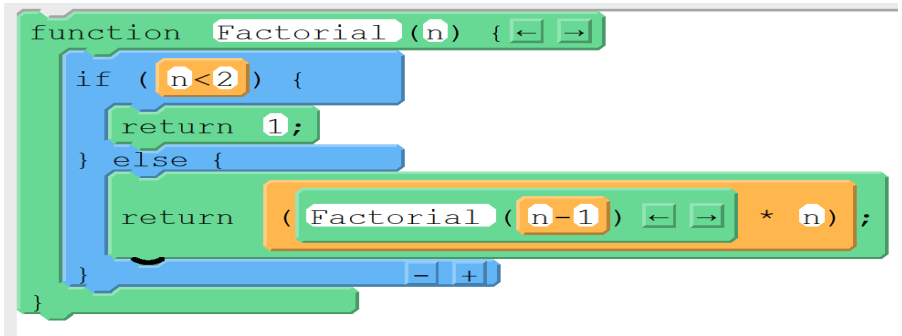


App Lab: build a "phone-like" app

Graphic editing of the App GUI (buttons, fields, labels, …)

Setters/getters of all App widgets properties

Full JavaScript-like visual syntax

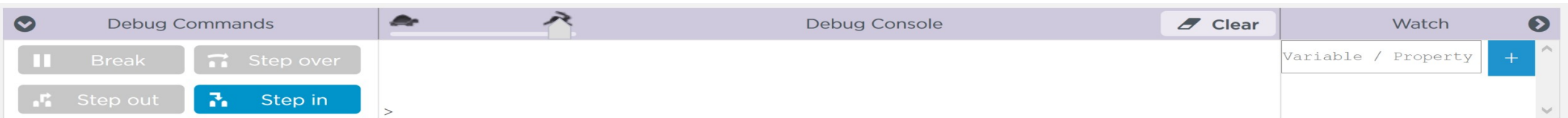


Full functions (args, local vars, return)

DATA store (dictionary OR tables)

Turtle graphics and Canvas

New: DEBUGGER!

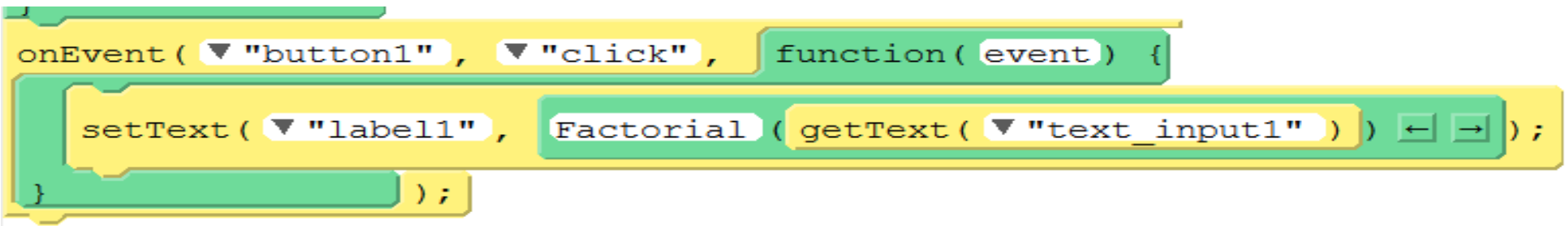


App Lab Events

Events:

GUI:	onEvent(widgetId, event, callback)
Data:	onRecordEvent(table, callback(record, event))
Timers:	setTimeout(ms, callback)
	timedLoop(ms, callback)

Callback functions (demo)



App Lab: custom libraries and datasets

You can export/import libraries of functions/blocks

Apart from remixing the teacher initial project, you can use/give to students external javascript libraries

You can export/import custom datasets

It's relatively easy to prepare data-analysis projects.

Students can either use open-data or collect data for further analysis

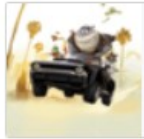
And many more …

Open-Ended Creativity


Sprite Lab


Dance Party


Poetry


The Bad Guys

Drawing


Artist


Frozen

Minecraft


Minecraft Adventurer


Minecraft Designer


Minecraft Hero


Minecraft Aquatic

Games with Events


Flappy


Star Wars (Blocks)


Bounce


Sports


Basketball

And many more …

Beyond Blocks


App Lab


Game Lab

Web Lab


Star Wars

Stories and Games with Play Lab


Play Lab


Infinity


The Amazing World of Gumball


Ice Age

Pre-reader


Play Lab (Pre-reader)


Artist (Pre-reader)