
# APPS II: Simulation

MESA CURRICULUM SERIES 2015
WASHINGTON MESA
1410 NE CAMPUS PARKWAY, SUITE 394, SEATTLE, WASHINGTON

# Apps II Curriculum:
## Simulation

*For Middle and High School*

## Introduction

In the past decade, the National Research Council has published major reports summarizing evidence from research studies that students learn best by experiencing and solving real-world problems. One of the most effective approaches to achieve this goal is Project-based Learning (PBL). Structured PBL experiences can provide a fertile environment for students to assume responsibility for their learning, shifting them from passive observers to more active learners. Effective PBL engages students in creating, questioning, and revising knowledge, while developing their skills in critical thinking, collaboration, communication, reasoning, synthesis, and resilience (Barron & Darling-Hammond, 2008).

Studies comparing learning outcomes for students taught via project-based learning versus traditional instruction show that when implemented well, PBL increases long-term retention of content, helps students perform as well as or better than traditional learners in high-stakes tests, improves problem-solving and collaboration skills, and improves students' attitudes towards learning (Strobel & van Barneveld, 2009; Walker & Leary, 2009). PBL raises expectations for learning for all students - particular student who are underrepresented in STEM. This approach awakens interests, increases motivation, and generates enthusiasm for learning science and engineering.

Washington Mathematics, Engineering, Science Achievement (MESA) builds a pathway to college and careers in science, technology, engineering, and mathematics (STEM) for students who are underrepresented in STEM fields, including African American, Native American, Latino, and female students. A leader in programs that support K-12 students to succeed academically and go to college. MESA takes a three-pronged approach to student achievement, focusing services on teachers, students, and parents to ensure student success.

MESA provides project-based enrichments and delivers teacher professional development that aligns with Washington State and national mathematics and science standards. Each MESA teacher attends at least 24 hours of professional development per year. MESA parents are involved in the education of their child with information, advocacy, and family math and science opportunities.

MESA serves schools and districts with high numbers of students that receive free or reduced lunch, are often in the most rural or the most urban areas, and have high enrollments of students from groups that are proven to be challenged by the largest opportunity gaps (OSPI website 1/2014).

## APPS II Curriculum

Washington MESA has responded to this need with project-based hands-on learning in computer science as one way to help close the opportunity gap. Beginning with APPS II curriculum, MESA has begun development of a comprehensive computer science (CS) program that will prepare students who have been underrepresented in STEM. This will prepare students with the critical thinking skills they need for the rigors of STEM disciplines especially in the computational science and computer science fields. In this curriculum, MESA students will receive foundational computing, and problem-solving skills through project based learning.

The curriculum addresses two of the many problems that must be addressed in any curriculum that is designed to increase academic achievement among underrepresented youth. The first is motivating students to participate regularly. The second is identifying teaching strategies, practices and, tools that can successfully engage students and increase academic achievement.

## Curriculum Summary

The curriculum presents computer science concepts to middle and high school students using Scratch, the visual programming platform from MIT. Through a series of hands-on lessons, the student will discover a fun a refreshingly new way of learning and applying mathematics and science concepts in their simulations. All the lessons are aligned with the State standards in science, mathematics and engineering and English Language Arts. The curriculum provides a quick look table with the sessions and the corresponding standards.

Students will spend 8-16 sessions using the online free software to complete the activities. After this, they will do brainstorming work to develop and complete their own simulations. There will a final showcase of the simulations when students will receive feedback from their peers to improve their work.

## Assessments

This curriculum contains formative assessments at the end of each session. A detailed rubric for peer and teacher feedback and an assessment rubric is provided for the final project. The students will work collaboratively in teams of up to four student to complete their projects.

## Credits

Thank you to everyone who has contributed the time to the review and pilot this curriculum. A special thanks to the curriculum writer whose skills; content knowledge and commitment were invaluable in the development of this curriculum for Washington MESA.

**Teacher Reviewers:** Pam Kidder and Cathy Beadle

**Curriculum Writer:** Peter Gruenbaum

**Layout:** Kuulani Seto and Ngoc Tien Do

**Curriculum Director:** Phyllis Harvey-Buschel

## Resources

Resources used in this curriculum include:

- Scratch: http://scratch.mit.edu
- Washington MESA Scratch projects: https://scratch.mit.edu/users/wamesa/
- ScratchEd (Scratch resources for teachers): http://scratched.gse.harvard.edu/
- Nobel Prize Educational Simulations: http://www.nobelprize.org/educational
- Bacterial growth video: http://www.cellsalive.com/ecoli.htm
- Padlet: http://padlet.com
- Edmodo: https://www.edmodo.com/

More detail on some of these resources can be found in the Resources section of the Overview.

## Further Information

For further information about Washington MESA enrichment curricula and teacher instructional support, contact:

Phyllis Harvey-Buschel
Curriculum and Teacher Instructional Support Director
pgharvey@uw.edu or 206-897-1714

For more information and Washington MESA, visit www.washingtonmesa.org.

## Overview

This curriculum is structured topically in an effort to focus students' learning around computer science ideas, and skills. It provides examples for teachers to use as a guide for students learning. Rather than focus on a single standard the curriculum uses the project-based approach to offer a more comprehensive learning experience for students. For teaching, this curriculum will provide a simple introduction to each lesson and allow the students to explore the topics more in-depth individually and in teams. Teachers are encouraged to allow students to brainstorm, have team discussions and help students develop their own understanding and ideas about the topic. Teachers will then guide them through the process of creating a simulation application and using visual programming language- Scratch.

**Grade level:** The curriculum is intended for use with **middle school** students **(6-8) and high school** students (9-12).

**Number of sessions:** There are **16** sessions in this curriculum. Each session begins with a summary, the grade level and provides a time frame for instruction. The standards table at the beginning of the document provides a snapshot of the connections of each lesson to the States' standards.

**Time per session:** The estimated time frame for each of the sessions will be about **50 minutes**.
*Note:* The lessons may be significantly shorter or longer than the estimated time, depending on how quickly the students learn. Teachers should always be prepared to move onto the next lesson in case the current lesson ends sooner than expected.

**Standards:** The Standards section describes how the curriculum is aligned with the Common Core English Language Arts and Next Generation Science Standards. Primarily, the curriculum meets several of the engineering components by having the students define the criteria and constraints of a problem that can be simulated. Students then designing the solution to that problem by developing a computer model, testing it, and improving it.

## Objectives:

The learning objectives are provided at the beginning of each lesson as a reminder of the performance outcome for students at the end of the unit. The students will be able to:

- Understand what a simulation is
- Analyze a simple simulation
- Create animations and simple games in Scratch
- Build simulations for an aquarium and gravity
- Work cooperatively in groups
- Design and build a simulation
- Test and improve a simulation

## Unit Overview

For this unit, teachers will be leading students through creating simulations using Scratch, which is a visual programming language. For the final project, the students will choose the kind of simulation they wish to create, which can be physical, biological, and/or economical. Their simulation must model a real-world situation and provide a visualization of the model.

Although this curriculum is designed for both middle and high school, there are several differences. Suggestions:

**Middle Schools**

- Students can do Lesson 2 and Lesson 3 as separate sessions. Students go through two sample simulations: the aquarium and the gravity simulation.
- Students are introduced to a third simulation, which is the lemonade stand. However, they do not build it.
- Students have 4 sessions to build their final project, whereas high school students only have 3.

**High Schools**

- For session 2, high school students may go through Lesson 2 and the first half of Lesson 3 as one session because the second half of Lesson 3 is a Scratch project that is too simple for high school students.
- Students start with the gravity simulation and then move on to the bacterial and energy simulations because the aquarium simulation may be too simple for high school students.
- Students may end up going faster through some of the earlier sessions, so it may be possible to move more quickly and provide more time to work on their projects in the end.

## Simulations

A simulation is a way of modeling something real in a way so that it somehow creates an illusion of being like the real thing. Simulations typically involve computers, but they do not have to.

A simulation has two parts:

- The *model* is a set of rules and equations that provide the math and logic for how the simulation will operate
- The *visualization* is what the user sees and interacts with, and typically uses computer graphics.

This curriculum will lead you through creating two to three simulations. For middle school, these are an aquarium (simple biology) and gravity (physics). For high school, these are gravity (physics), bacterial growth (biology), and power plants (energy and economics). These are very different types of simulations that should give teachers and students the sense of the variety of simulations that are possible. Then the students will brainstorm their own ideas.

## Scratch

Scratch is a visual programming language that was developed at MIT. It works extremely well for middle school. Instead of using text, it uses a series of blocks that are shaped like jigsaw puzzle pieces that snap

together. This means that students do not need to learn computer programming language syntax, which is often frustrating for this grade level. However, Scratch introduces students to many computer science concepts that are used in real programming languages, such as loops, conditionals, variables, and events.

Scratch is web-based, so no software installation is required. Simply go to the web address http://scratch.mit.edu. If you sign up for an account, once you are logged in, all projects are automatically saved, so there is no need to remind students to save their work.

**Note:** Adobe Flash is required for Scratch to run. This means it will not run on iPads. There is an iPad app alternative called Pyonkee which is the equivalent of Scratch, but it is an earlier version that does not look exactly like this one, although it has much of the same functionality. Unfortunately, Pyonkee does not have the cloning feature, which is required for the high school curriculum.

There is a lot of functionality in Scratch, and if you are not familiar with programming, you may find it difficult that you do not understand everything about it. Also, you will find that some students outstrip your learning and figure out how to do things that you don't know. To teach software often means giving up the idea that you know more than the students. Here are some teaching strategies you can use:

- Teach the skill of looking up how to do things. There is a lot of information about Scratch online, including video tutorials.
- At the end of each class with Scratch, ask what students have learned. Keep a list publically available where students are listed as being experts on how to do certain things. Add their skills to the list. If a student cannot figure out how to do something, look up on the list to see if there is an expert you can send that student to.

## Alternatives to Scratch

If, for whatever reason, Scratch is not the best tool for your class, you are welcome to use others. You will need to adapt the programs in the curriculum for the platform that you use. Here are some alternatives to consider:

- AppInventor from MIT. A block-based visual language that runs on Android devices. It is similar to Scratch, but more sophisticated and advanced, and not as easy to create graphics.
- TouchDevelop from Microsoft. A text-based visual language that is a hybrid of text languages and visual languages. It can run on almost any device.
- NetLogo from Northwestern University. A visual simulation language program. Can run very sophisticated simulations, but teaches less about computer science concepts.
- Text-based programming languages like Java, Python, etc. These are good for very advanced students, but require more computer science knowledge.

## Students of Different Abilities

One of the most difficult aspects of teaching coding in a project-based setting is that some students pick it up much more quickly than others. As you lead students through the initial projects, some will finish

far before others have. These students may then get bored and start doing things on the computer that are off task.

You want to encourage the students who are ahead to challenge themselves to do more with Scratch. Many of the sessions end with a section called "For Advanced Students". This is a short challenge they can do if they finish the assignment faster than others.

Students who are so advanced they are ready for something beyond Scratch might want to try TouchDevelop by Microsoft. TouchDevelop is a hybrid of a visual-based programming language (like Scratch) and a text-based programming language (like Java). TouchDevelop can run on any device.

## Lessons

This unit is organized into lessons. Typically, a lesson corresponds to one 50 minute session, but some span multiple sessions. Most apply to both middle and high school, but some are just for one of the other.

**Lesson 1:** **What is Simulation?** Explains what the unit will cover and discusses what simulation is and how it is used. A sample simulation is demonstrated.

**Lesson 2:** **Simulation Model and Visualization.** Explains how a simulation consists of a model (logic and equations) and a visualization (graphics and interaction). Students try out a simulation and the class determines the model and visualization.

**Lesson 3:** **Introduction to Scratch.** Explains how to get started in Scratch. Middle school students build a simple animation.

**Lesson 4:** **Computer Science Concepts in Scratch.** Introduces students to several computer science concepts by building a project where you move a sprite around with the arrow keys. Concepts include the x/y coordinate system, loops, and conditionals.

**Lesson 5:** **Variables and Random Numbers.** Introduces students to two important computer science concepts: variables and random numbers. Students build on their previous project to create a simple game that places a sprite in a random position and uses a variable to keep track of the score.

**Lesson 6:** **Aquarium Simulation.** For middle school only. Students model a simple system of two aquariums and three sea creatures. They also learn about sprite direction.

**Lesson 7:** **Gravity Simulation.** Students learn how gravity is modeled and then create a simple simulation where a sprite falls. They finish by making a project where a sprite jumps.

**Lesson 8:** **Bacteria Simulation 1.** High school only. Students learn about modeling bacterial growth and how to clone sprites.

**Lesson 9:** **Bacteria Simulation 2.** High school only. Students build a bacteria simulator in Scratch and use it to plot growth for two scenarios.

**Lesson 10:** **Energy and Economics Simulation.** High school only. Students analyze a simple simulation of coal plants and wind farms that models cost, energy, and carbon dioxide production. Then they use the web to research how to add a natural gas plant to the simulation.

**Lesson 11:** **Choosing a Simulation Project.** Students form groups and brainstorm a simulation project. Middle school students are introduced to a simple economic simulation game, which is a lemonade stand.

**Lesson 12:** **Creating the Model.** Student groups create a model for their simulation.

**Lesson 13:** **Creating the Visualization Design.** Student groups create a visualization design for their simulation.

**Lesson 14:** **Building the Simulation.** Students have 3 to 4 sessions in which to build their simulation.

**Lesson 15:** **Testing and Improving.** Students will test each other's simulations and provide feedback. Then the groups will work on making fixes and improvements.

**Lesson 16:** **Final Model and Presentation.** Students will revisit their original model and update it to reflect their simulation, since changes are likely to have occurred. Then they will present both their model and their simulations to the class.

## Resources

This section contains some useful resources to help with this curriculum.

### Scratch

There are many resources online for learning Scratch. In particular, scratchEd has many resources for teachers. If time permits, you may want to review these resources on your own prior to the class.

Throughout the curriculum, there are links to Scratch projects. All of these projects, plus projects that other MESA teachers have created, are available at the Washington MESA Scratch account website: https://scratch.mit.edu/users/wamesa/.

### Collaboration Tools

There are tools available that make it easier for your students to collaborate with you and their fellow students. These tools allow you and your students to easily organize and share documents and resources on the web. Two tools you may want to investigate are:

- padlet at http://padlet.com
- Edmodo at https://www.edmodo.com/

## Standards

This section maps the lessons to the State standards. The standards are the Common Core English Language Arts and mathematics and The Next Generation Science Standards. The standards are first described and then there is a table below that which standards apply to each lesson.

### Common Core and Next Generation Science Standards 6-12: Engineering Design

**MS-ETS1-1:** Define the criteria and constraints of a design problem with sufficient precision to ensure a successful solution, taking into account relevant scientific principles and potential impacts on people and the natural environment that may limit possible solutions.

**Science and Engineering Practice:**

- Define a design problem that can be solved through the development of an object, tool, process or system and includes multiple criteria and constraints, including scientific knowledge that may limit possible solutions.

**MS-ETS1-2:** Evaluate competing design solutions using a systematic process to determine how well they meet the criteria and constraints of the problem.

**Disciplinary Core Idea:**

- There are systematic processes for evaluating solutions with respect to how well they meet the criteria and constraints of a problem.

**MS-ETS1-3:** Analyze data from tests to determine similarities and differences among several design solutions to identify the best characteristics of each that can be combined into a new solution to better meet the criteria for success.

**Disciplinary Core Idea:**

- There are systematic processes for evaluating solutions with respect to how well they meet the criteria and constraints of a problem.
- Sometimes parts of different solutions can be combined to create a solution that is better than any of its predecessors.

**MS-ETS1-4:** Develop a model to generate data for iterative testing and modification of a proposed object, tool, or process such that an optimal design can be achieved.

**Disciplinary Core Idea:**

- A solution needs to be tested, and then modified based on the test results, in order to improve it.
- Models of all kinds are important for testing solutions.

**HS-ETS1-1:** Analyze a major global challenge to specify qualitative and quantitative criteria and constraints for solutions that account for societal needs and wants.

**Science and Engineering Practice:**

- Analyze complex real-world problems by specifying criteria and constraints for successful solutions. (HS-ETS1-1).

**HS-ETS1-2:** Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering.

**Disciplinary Core Idea:**

- Criteria may need to be broken down into simpler ones that can be approached systematically, and decisions about the priority of certain criteria over others (trade-offs) may be needed.

**HS-ETS1-3:** Evaluate a solution to a complex real-world problem based on prioritized criteria and trade-offs that account for a range of constraints, including cost, safety, reliability, and aesthetics as well as possible social, cultural, and environmental impacts.

**Disciplinary Core Idea:**

- When evaluating solutions, it is important to take into account a range of constraints, including cost, safety, reliability, and aesthetics, and to consider social, cultural, and environmental impacts.

**HS-ETS1-4:** Use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem.

**Disciplinary Core Idea:**

- Both physical models and computers can be used in various ways to aid in the engineering design process. Computers are useful for a variety of purposes, such as running simulations to test different ways of solving a problem or to see which one is most efficient or economical; and in making a persuasive presentation to a client about how a given design will meet his or her needs.

## Common Core and Next Generation Science Standards 6-12: Life Sciences

**MS-LS1-3:** Use argument supported by evidence for how the body is a system of interacting subsystems composed of groups of cells.

**MS-LS1-8:** Gather and synthesize information that sensory receptors respond to stimuli by sending messages to the brain for immediate behavior or storage as memories.

**MS-LS2-3:** Develop a model to describe the cycling of matter and flow of energy among living and nonliving parts of an ecosystem.

**HS-LS1-2:** Develop and use a model to illustrate the hierarchical organization of interacting systems that provide specific functions within multicellular organisms.

**HS-LS2-1:** Use mathematical and/or computational representations to support explanations of factors that affect carrying capacity of ecosystems at different scales.

**HS-LS2-7:** U  Design, evaluate, and refine a solution for reducing the impacts of human activities on the environment and biodiversity.

## Common Core and Next Generation Science Standards 6-12: Physical Sciences

**MS-PS2-4:** Construct and present arguments using evidence to support the claim that gravitational interactions are attractive and depend on the masses of interacting objects.

**HS-PS2-4:** Use mathematical representations of Newton's Law on Gravitation and Coulomb's Law to describe and predict the gravitational and electrostatic forces between objects.

## Common Core Standards for Mathematical Practice

**CCSS-7.EE.3:** Solve multi-step and real-life problems.

**CCSS-HSP-LE.A.2:** Construct and compare linear, quadratic, and exponential models and solve problems.

## Common Core Standards for English Language Arts

**CCSS.ELA-LITERACY.W.9-10.2.A:** Introduce a topic; organize complex ideas, concepts, and information to make important connections and distinctions; include formatting (e.g., headings), graphics (e.g., figures, tables), and multimedia when useful to aiding comprehension.

**CCSS.ELA-LITERACY.SL.11-12.5:** Make strategic use of digital media in presentations to enhance understanding of finding, reasoning, and evidence and to add interest.

## Career and Technical Education (CTE) Washington State Standards (21st Century Skills)

**Learning & Innovation Skills:** Skills that prepare for a more complex life, work environment, and are essential to prepare for the future.

- Critical Thinking and Problem-Solving
- Communication and Collaboration
- Creativity and Innovation

**Information, Media & Technology Skills:** Ability to exhibit a range of functional and critical thinking skills related to information, media and technology.

- Information Literacy
- Media Literacy
- Information, Communication, and Technology (ICT) Literacy

## Mapping Lessons to Standards

The following table shows which standards apply to each lesson, either in general and/or the relevant Science and Engineering Practices (SEP) and Disciplinary Core Ideas (DCI).

| Standards | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **MS-ETS1-1** | | | ● | | | | ● | ● | | | | ● | ● | | ● | ● | |
| | SEP: Define a design problem | | | | | | ● | ● | | | | ● | ● | | ● | | |
| **MS-ETS1-2** | DCI: Systematic processes for evaluation | | | | | | | | | | | ● | | | | ● | |
| **MS-ETS1-3** | DCI: Systematic processes | | | | | | | | | | | ● | | | | ● | |
| | DCI: Combine solutions | | | | | | | | | | | ● | ● | | | | |
| **MS-ETS1-4** | | | | | | | ● | ● | | | | ● | | | | | |
| | DCI: Solutions need to be tested | | | | | | | | | | | | | | ● | ● | |
| | DCI: Models are important for testing | | | | | | ● | ● | | | | | ● | | ● | ● | |
| **HS-ETS1-1** | | | | | | | | | ● | ● | ● | ● | ● | | | | |
| | SEP: Analyze real-world problems | | | | | | | | ● | ● | ● | ● | ● | | | | |
| **HS-ETS1-2** | | | | | | | | | | | ● | ● | ● | ● | | | |
| | DCI: Criteria may need to be broken down | | | | | | | | | | ● | ● | ● | ● | | | |
| **HS-ETS1-3** | | | | | | | | | ● | ● | ● | ● | | | | ● | |
| | DCI: Range of constraints | | | | | | | | ● | ● | ● | ● | | | | ● | |
| **HS-ETS1-4** | | ● | ● | | | | | ● | ● | ● | ● | ● | ● | ● | ● | ● | |
| | DCI: Computers | ● | ● | | | | | ● | ● | ● | ● | ● | ● | ● | ● | ● | |
| **MS-LS1-3** | | ● | | | | | | | | | | | | | | | |
| **MS-LS1-8** | | | ● | | | | | | | | | | | | | | |
| **MS-LS2-3** | | | | | | | ● | | | | | | | | | | |
| **HS-LS1-2** | | ● | | | | | | | | | | | | | | | |
| **HS-LS2-1** | | | | | | | | | ● | ● | | | | | | | |
| **HS-LS2-7** | | | | | | | | | | | ● | | | | | | |
| **MS-PS2-4** | | | | | | | | ● | | | | | | | | | |
| **HS-PS2-4** | | | | | | | | ● | | | | | | | | | |
| **CCSS-7.EE.3** | | | | ● | ● | ● | | | | | | | | | | | |
| **CCSS-HSP-LE.A.2** | | | | | ● | ● | | | | ● | | | | | | | |
| **CCSS.ELA-LITERACY.W.9-10.2.A** | | | | | | | | | | | | | | | | | ● |
| **CCSS.ELA-LITERACY.SL.11-12.5** | | | | | | | | | | | | | | | | | ● |
| **Learning & Innovation Skills** | | | | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| **Information, Media & Technology Skills** | | ● | ● | | | | | | | | ● | ● | ● | | ● | | ● |

## Rubrics

To help with rubrics, we provide some suggestions on how to use the Rubistar website, which assists teachers in creating rubrics for project-based learning activities. Each Rubrics section contains a link to a page to help generate a rubric, as well as recommendations on what rubric categories to add to the rubric. Note that there are no exact matches for simulation or coding projects, so for each lesson, we chose a project type that had relevant rubric categories to choose from.

# Lesson 1: What is Simulation?

Middle school: Session 1
High school: Session 1

## Summary

Students learn what the unit will cover and then are given an introduction to simulation by watching an interactive simulation about blood typing and transfusions. Finally, they discuss what simulation is and how it is used.

## Learning Objectives

After doing this session, students should be able to:

- Describe the objective of the unit
- Learn what simulation is and how they are used

## Materials

- Teacher's computer with internet
- Projector

## Preparation

Before class, review the material to cover and try out the blood type simulation as described in the **Lesson Procedures** section.

## Vocabulary

The following vocabulary is used in this lesson:

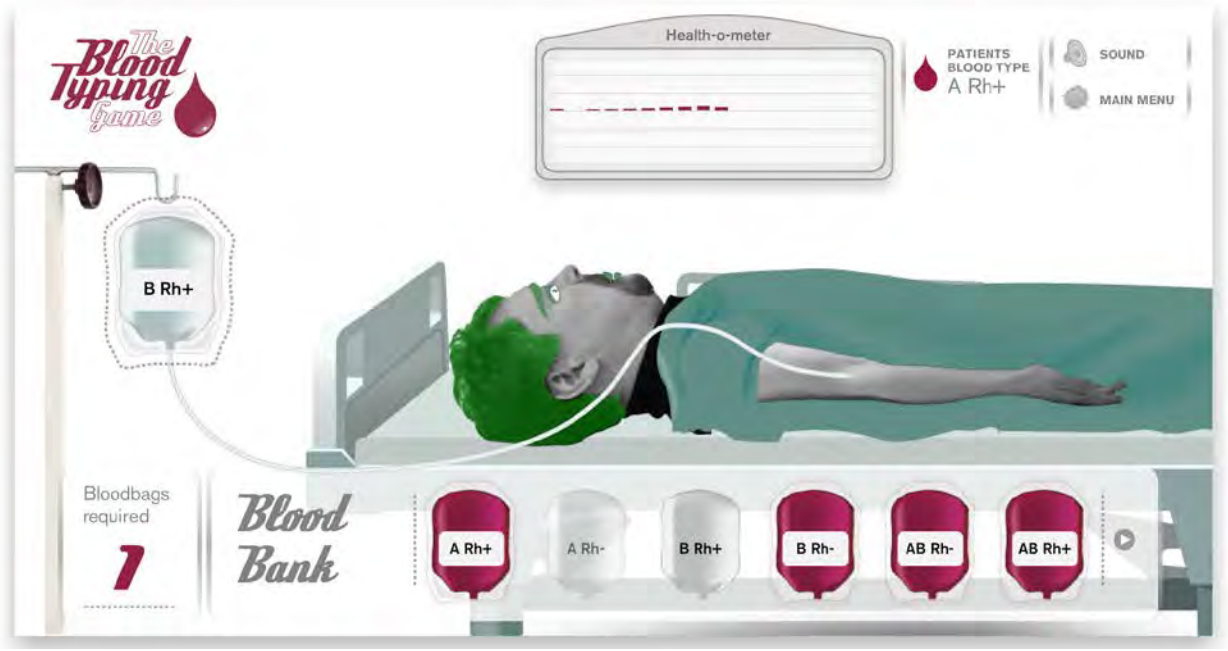
- **Simulation** – The representation of the behavior or characteristics of one system through the use of another system, especially a computer program designed for the purpose.

## Introduction

Explain what this unit will cover, which is:

- Learning what a simulation is
- Analyzing a simple simulation
- Learning Scratch
- Building simulations for
    - Middle School: aquarium, gravity
    - High School: gravity, bacteria, and energy production
- Working in groups to come up with their own simulation
- Designing and building your simulation
- Testing and improving your simulation
- Presenting your simulation to the class

## Lesson Procedure




Follow these steps to show your students a simulation for determining people's blood types and giving them blood transfusions:

1. Explain that this simulation first has you draw blood from a car crash patient and then analyze it. Then, given the blood type, it has you choose what kinds of blood to use in a transfusion.
2. Explain how people can have blood type A, B, AB, or O. (O blood has neither A nor B.)Their blood can also be Rh positive or negative. If you give a transfusion with blood that has A or B, and the patient's blood does not have that, they will have a bad reaction. For example, you can give A blood to either an A or AB patient, but not a B or O patient. You can give O blood to any patient. Similarly, you can give Rh- blood to Rh+ or Rh- patients (because Rh- doesn't have Rh), but you can only give Rh+ blood to Rh+ patients. If you give Rh+ blood to a patient that does not have Rh (Rh-), they will have a bad reaction.
3. Go to http://www.nobelprize.org/educational/medicine/bloodtypinggame/. Note that Adobe Flash is required for your browser.
4. Click on **Play the Blood Typing Game**.
5. Click on **Proceed** once the game is loaded.
6. Click on **Quick game – same patients** (the last card.)
7. Explain the game and then click on **Proceed.**
8. Select the first patient.
9. Drag and drop the syringe on the patient's arm.
10. Drag and drop the syringe onto the test tube labeled A.
11. The liquid in the test tube stays red, which means that it is not a blood type that contains A.
12. Drag and drop the syringe onto the test tube labeled B.

13. The liquid in the test tube stays red, which means that it is not a blood type that contains B. It must therefore by blood type O.
14. Drag and drop the syringe onto the test tube labeled Rh.
15. The liquid in the test tube turns clear, which means that it is Rh positive.
16. Click on O to show that it is type O. Click on Rh+ to show it is Rh positive.
17. Close the "You're bloody right" message.
18. This patient requires one bag of blood. You need to find the one that is labeled O Rh+. Click on the arrow on the right to find it. Then drag it up to hang it on the hook.
19. Select the next patient (green hair). Do the same procedure to find the blood type. This one you will find is A Rh+.
20. This patient needs two bags of blood, but there is only one A Rh+ bag. After doing the A Rh+ bag, try giving them a B Rh+ bag. That got a negative reaction! It turns out that an A Rh+ patient can also accept A Rh- blood, or any type of O blood. Drag any of those bags in to finish the level.
21. If there is time, do the third patient.

## Practice Activity

After the students have watched the simulation, break students up in small groups (3-4 students) to have a discussion about what they learned from the simulation.

Allow 6-8 minutes for each student in the group to share what they learned. Explain that each student will have 2 minutes only to share. Have the students choose one person to share with the whole class a summary of what they learned. Use the following questions as a guide for the small group discussion and reporting out:

- What is a simulation?
- Why do you think people use simulations?
- Can you name other simulations you have seen or used?

Once you have heard from all the groups, here are some possible answers you could bring up:

- Simulation is something that looks and acts like something, but is not real.
- Simulation is used for practice (flight simulators, surgery), entertainment (SIM games), research (scientific simulation)
- Simulations they might know about: flight, driving, boats, surgery, architecture, engineering, military, games (SIM games, Farmville, Clash of Clans)
- Simulations often use computers, but they don't have to

## Rubric

For the groups sharing out, use the Rubistar Oral Presentation rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=4&section_id=1&

Use the rubrics: Comprehension, Listens to other Presentations, Stays on Topic, and Content

# Lesson 2: Simulation Model and Visualization

Middle school: Session 2
High school: Session 2

## Summary

Describe how simulation consists of a model (logic and equations) and a visualization (graphics and interaction). Discuss the concept of conditioned reflexes, as illustrated by the Pavlov dogs experiment. Students play a simulation game that allows them to train a dog to respond to sounds. Then, as a class, you lead them through a discussion to determine the visualization and the model.

**Note:** For high school, session 2 is continued in the next section.

## Learning Objectives

After doing this session, students should be able to:

- Describe what a simulation model
- Describe what a simulation visualization is
- Determine what the visualization and model are for simple simulations

## Materials

- A computer for each student with internet connection and browser
- Projector or whiteboard to display URL for students to copy

## Preparation

Before class, review the lesson and try out the Pavlov dog simulation at: http://www.nobelprize.org/educational/medicine/pavlov/. To win the game, you should make a noise and then quickly feed meat to the dog. Use the same noise and meat three times in a row to create a conditioned response.

## Vocabulary

The following vocabulary is used in this lesson:

- **Model** – The math and logic behind a simulation.
- **Visualization** – The graphics that show the results of the model and sometimes allow users to interact with the model.
- **Conditioned Reflex** – A learned action that someone does without thinking about it.

## Introduction

In this session, you will teach about how simulations have a model (the math and logic) and a visualization (the graphics and user interaction). Students will try out a simulation and then work with you to define the model and the visualization.

## Model and Visualization

Start by reviewing what a simulation is and why they are important (information from the last session). Then explain that a simulation has two parts:

- The *model.* This is the math and logic behind the simulation. It provides the rules to how the simulation works. The better the model, the more it will be like the real thing. The model is not directly visible to the user.
- The *visualization.* The visualization takes the model and makes it visible to the user. It consists of graphics that show the simulation results, and, if the simulation is interactive, it provides the interaction with the user.

In the blood type simulation, the model was the rules about how different blood types respond to the tests in the test tubes, and what kinds of blood different patients can accept, depending on their blood type. The visualization was the images of the patients, syringes, test tubes, etc., and included the ability to move the syringe around, choose the blood type, and drag blood bags in place.

Both the model and the visualization require creating software code, and students will be doing both in this class.

## Conditioned Reflexes

Explain to the students that they will play with a simulation in a moment, but that you want to first tell them what it is about. Many animals, including humans, have what is called “conditioned reflexes”. A reflex is an action that you take without thinking about it. For example, if someone waves their hand near your eyes, you automatically blink. That is a reflex.

You were born with the blinking reflex. However, a conditioned reflex is one that you learn. About a hundred years and twenty years ago, a scientist named Pavlov did an experiment where he would ring a bell just before feeding a dog. After he did this several times, he noticed that the dogs would salivate (drool spit) just before feeding. Soon, he found that he could ring the bell, and it would make the dogs salivate, even if there was no food in sight. The salivation was a reflex, and it was learned. For more information on Pavlov’s experiment, see http://www.simplypsychology.org/pavlov.html.

Discuss this concept with the class. Can they think of examples of where people react without thinking based on behavior they have learned? For example, stopping at a red light, or checking their phone when they hear a sound that means they got a text.

## Lesson Procedure

### Play Simulation


Have the students play the Pavlov's Dog simulation at http://www.nobelprize.org/educational/medicine/pavlov/. They will need to consistently choose a sound and immediately follow it up with food several times in order to get the dog to salivate when just hearing the sound. See if they can figure out how many times that is.

### Practice Activity

Now, as a class, discuss what the visualization is. It should include elements like:

- The animated dog
- The food
- The noisemakers
- The ability to play a noise by clicking on a noisemaker
- The ability to drag food into the bowl

Next, discuss the model. This will be harder for the students. See if you can lead them to come to these set of rules:

- To train the dog, you must make a noise and follow it immediately by feeding it meat.
- You must give meat to the dog within 3 seconds or the dog will fall back asleep
- If you feed the dog bananas, it doesn't count
- The noisemaker must always be the same
- The meat must always be the same
- If you do this 3 times, then the next time you make that noise, the dog will salivate

## Rubric

For the class discussion, use the Rubistar Oral Presentation rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=4&section_id=1&

Use the rubrics: Comprehension, Listens to other Presentations, Stays on Topic, and Content

# Lesson 3: Introduction to Scratch

Middle school: Session 3
High school: Session 2 continued. Skip Project section.

## Summary

In this unit, students will use MIT's visual programming language Scratch to build their simulations. This class is an introduction to how Scratch works. Students will build a simple animation to get some practice with it.

**Note:** For high school, only cover the introduction to Scratch piece of this section. Skip the project, since it is too simple for them. They will get hands-on experience in the next session.

## Learning Objectives

After doing this session, students should be able to:

- Learn how to log into Scratch
- Build a simple project in Scratch (middle school only)
- Share a project so that it is publically available

## Materials

- A computer for each student with internet
- Teacher's computer with internet
- Projector
- Email accounts that all students can use.

**Note:** If students do not have school email accounts, then you may need to create several email accounts for your class with a service such as Gmail. Alternatively, you can create one email account for the class and have all of the students use it.

## Preparation

Go through the steps in the **Lesson Procedure** section and make sure you understand how to do them.

## Vocabulary

The following vocabulary is used in this lesson:

- **Sprite** – In Scratch, collection of computer code, graphics, and sound.
- **Script** – A series of instructions in the form of computer code.

## Introduction

In this session, you will help students get familiar with Scratch. Discuss Scratch and how it is a visual way to program computers to create games and animations. Most computer programs require you type in text, but Scratch lets you drag and drop jigsaw puzzle pieces to create code. The shapes of the puzzle pieces give you clues as to how they fit together. Although easy to use, Scratch is a real programming

language and students will be learning concepts that college students learn in computer programming classes.

Some of the students might have used it before, so be sure to ask before starting. Have these students be the experts that other students can go to with questions.

## Lesson Procedure

### Accounts

To save your work in Scratch, you must have an account. You can have individual accounts for all of the students, or you can have one account for the entire class. Just make sure you set up a system where if students forget their name and password that they will not lose all of their work. You must have a valid email for all accounts, so if this is an issue for students, then just set up one account for the class.

Have students follow these steps to create a new account:

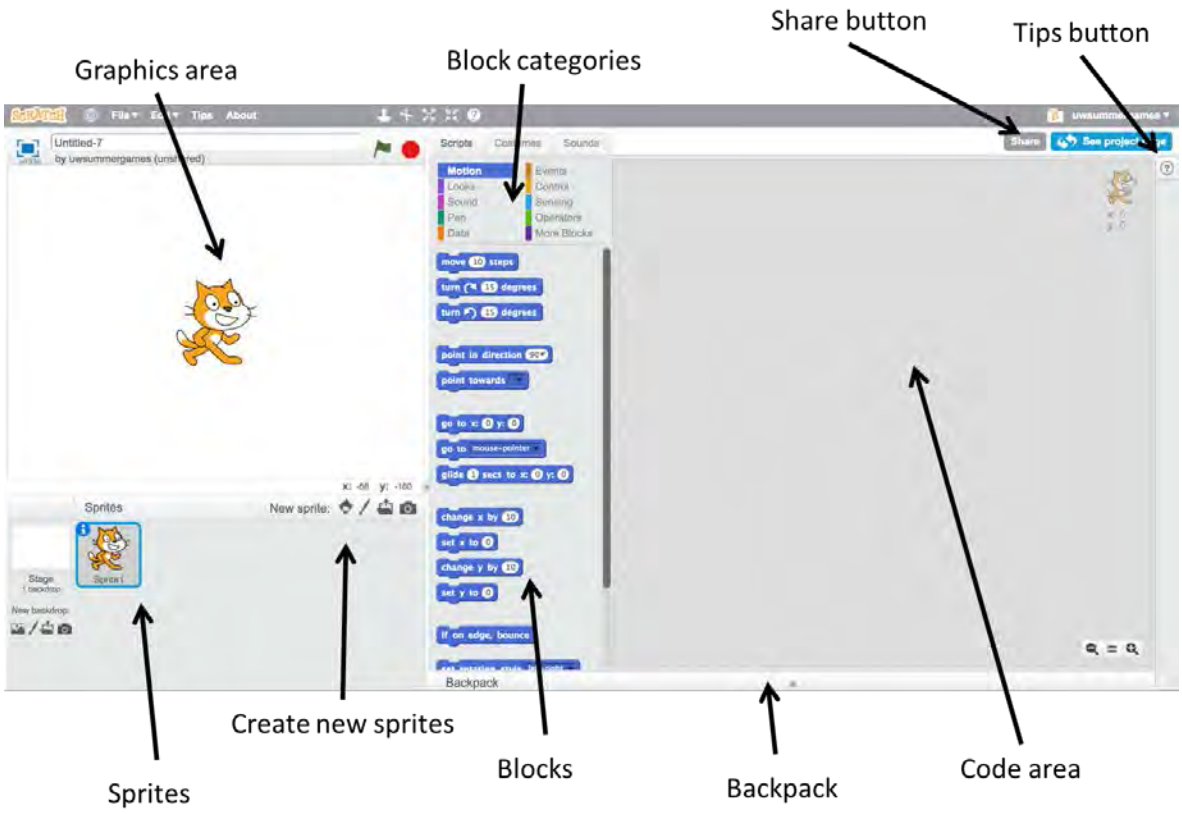
1. In a browser, navigate to http://scratch.mit.edu.
2. Click on **Join Scratch**.
3. Enter a username and password. Click **Next**.
4. Enter Birth month and year, gender, and country. Click **Next**.
5. Enter the email address. Click **Next.**
6. You are now ready to start using Scratch. Be sure to verify the account through the email sent to your email address.

In the future, to sign into an account, click on the **Sign In** button.

### User Interface

Give your students an overview of the Scratch User Interface.

1. Click on the **Create** button. This will create a new project.
2. Point out the various pieces of the user interface, as shown in the diagram below. You may have to work with Scratch a little before understanding what all of these pieces mean.

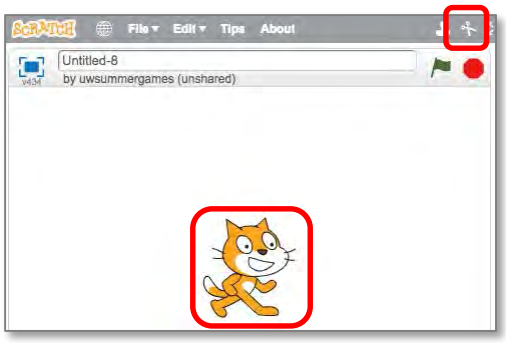



a. The graphics area is where you put sprites as part of your simulation visualization. You can also choose a background.
b. The block categories contain parts of code. For example, **Motion** contains all the blocks that control how sprites move around the screen and **Looks** contains all the blocks that control what your sprite looks like.
c. The share button is used to make your project public. Once public, other people can try it out, and can also "remix" it, meaning that they can take your code and add to it and change it.
d. The tips button either shows or hides the tips panel. The tips panel is useful for getting information on how to do to common tasks.
e. The sprites area shows all of the sprites in the project. The little blue **i** at the top left corner of each sprite button allows you to change details about the sprite, such as its name and how it rotates.
f. There are several buttons for creating sprites. Sprites can be chosen from a library, drawn by hand, or imported from a file.
g. For each category of block selected, you can choose from a number of blocks. Drag them to the code area to use.
h. The "backpack" is a place where you can drag code or sprites to make it easier to transfer between projects.
i. The code area is where you place your blocks of code for a sprite.

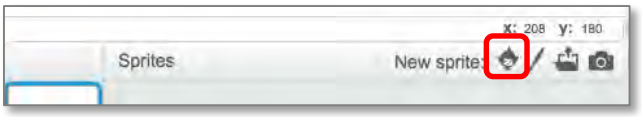
## Project (Middle school only)

Demonstrate creating a simple animation in Scratch: two characters telling a joke. Have the students watch, but not follow along on their computers.

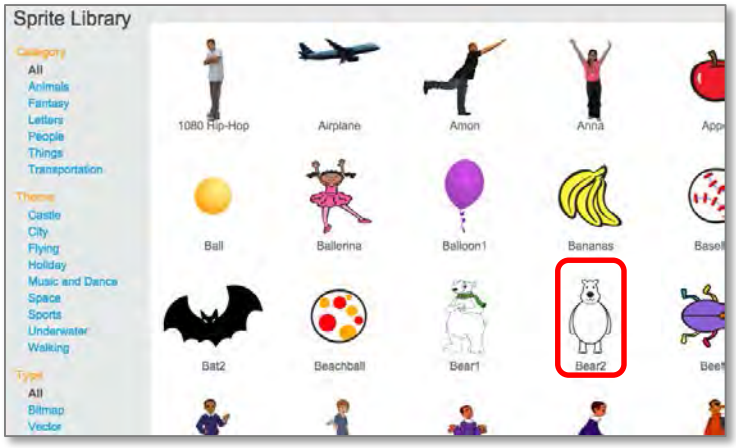
1. Remove the cat by clicking on the scissors icon at the top, and then clicking on the cat.



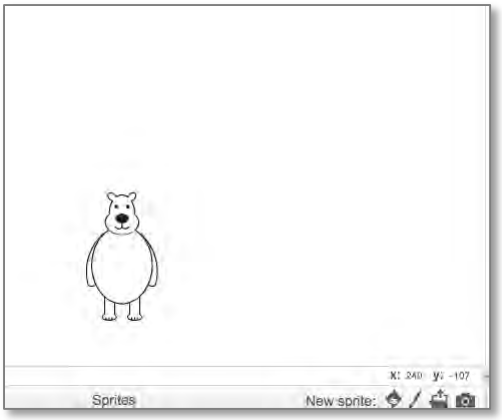

2. Next, you'll add two sprites from the library. Choose a new sprite from the library by clicking the first icon in the **New sprite** list.



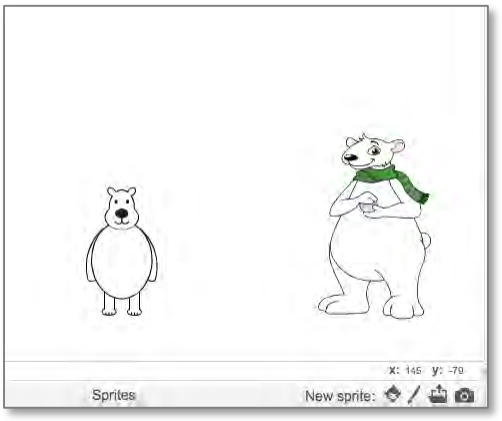

3. Choose the Bear2 image.



4. Drag the bear to the left half of the screen.

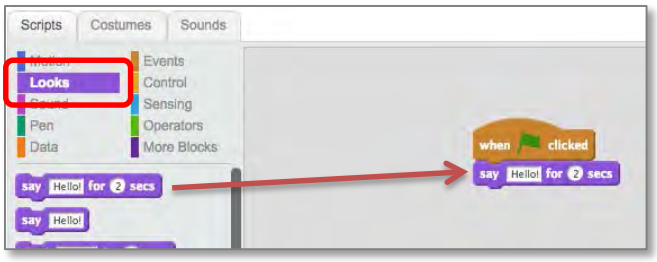



5. Repeat steps 2 and 3, but this time choosing Bear1. Drag that bear to the right half of the screen.




6. Now, you will need to add the code for the dialog. In the **Scripts** tab, click on **Events**. Drag the block called “When green flag clicked” out into the code area. This starts a series of steps that will happen when the game is started (that is, when the green flag is clicked).

7. Next, click on **Looks**. Drag the block called “Say Hello! For 2 secs” and put it directly under the “When green flag clicked”. It should snap in place like a jigsaw puzzle.




8. Click on “Hello!” and change it to “Why was 6 afraid of 7?”


9. Now, click on the green flag to start the animation. You will see the bear say, “Why was 6 afraid of 7?”

10. Next, in the **Sprite** section, click on the other bear (Bear2). This will allow you to create scripts for the other bear sprite.


11. Click on **Events** and then drag "When green flag clicked" out to the code area. Then click on **Control** and drag out "Wait 1 sec". Click on the 1 and change it to 2, so that this bear waits two seconds before doing anything. Finally, click on **Looks**, and drag out "Say Hello! for 2 secs." Click on "Hello" and change it to "Why?" Your script should look like this:


12. Now click the green flag. You'll see the two bears having a dialog.
13. Click on Bear1 again and add "wait **2** secs" (from the **Control** category) and then "say **Because 7 ate 9** for **2** seconds (from the **Looks** category). Remember that you will have to click and change the number of seconds and the dialog. Your Bear1 script should look like this:


14. Click the green flag to watch the entire animation.
15. Click the Share button at the top right to share the project. Now you can give the web address in the browser (URL) to anyone, and they can play the animation.


16. Finally, have the students make their own animation with dialog. This joke format is an easy one to use. The steps should be simple enough that they remember how you did it.

**For advanced students:** If students are already familiar with Scratch, they can add movement to their sprites.

## Practice Activity

At the end of class, ask students:

- How do you log into Scratch?
- Point to parts of the Scratch web page and ask students to explain what it does
- Ask the students to demonstrate how to share a project in Scratch?

## Rubric

For the Scratch animation, use the Rubistar Storyboard - multimedia rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=2&section_id=3&

Use the rubrics: Content and Required Elements

# Lesson 4: Computer Science Concepts in Scratch

Middle school: Session 4
High school: Session 3

## Summary

Students learn computer science concepts in Scratch, including sprites (images and code grouped together), coordinates (x/y coordinate system), loops (repeated instructions), and conditionals (statements that use "if… then…"). They learn these by building a project that moves a sprite around the screen using arrow keys.

## Learning Objectives

After doing this session, students should be able to:

- Create a new sprite
- Understand the x/y coordinate system and how it applies to Scratch
- Create code that repeats (loops)
- Create code that handles conditions (if-then blocks)
- Create a sprite that moves around with the arrow keys

## Materials

- A computer for each student with internet
- Teacher's computer with internet
- Projector
- Whiteboard or equivalent

## Preparation

Read through the **Explanation** sections and make sure you understand them. Go through the steps in the **Hands-on Work** sections and make sure you can do them.

## Vocabulary

The following vocabulary is used in this lesson:

- **Sprite** – In Scratch, collection of computer code, graphics, and sound.
- **Coordinate System** – A method of describing a position on a screen in terms of horizontal numbers (x) and vertical numbers (y).
- **Loop** – A block of instructions that is repeated.
- **Conditional** – A block of instructions that is only executed if a condition is true. Also known as an "if/then" block.
- **Event** – The start of a block of code, which occurs when something particular happens.

## Introduction

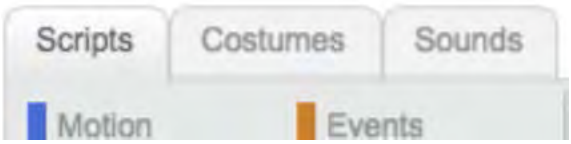
In this session, you will lead students through creating a sprite that moves around the screen using the arrow keys. There are four concepts you will cover: sprites, coordinate system, loops, and conditionals. For each of these, you will give a short explanation and then have the students work on something on Scratch. While you are demonstrating what to do, make sure the students have their hands off of the computer.

The goal of this class project is to create sprite that can move around with arrow keys. In order to finish in the 50-minute timeframe, each section should only take about 10 minutes.

## Sprites

### Explanation

Explain to the students that sprites are objects that contain instructions, graphics, and sound. The instructions are called "scripts", the graphics are called "costumes", and the sounds are called "sounds". You can see the three tabs that let you access those three pieces.




### Hands-on Work

We learned from last week how to create a sprite from the sprite library. Demonstrate how to create a new sprite and then shrink it down. Follow these steps:

1. From the **File** menu, choose **New** to create a new project.
2. Choose the scissors and click on the cat to remove the cat.
3. From the **New sprite** area, choose library. 
4. Choose a character, such as the bat.
5. Click on the shrink button at the top, which is four arrows pointing inwards.


6. Click on the sprite several times to shrink it to about 3/4 of its size.

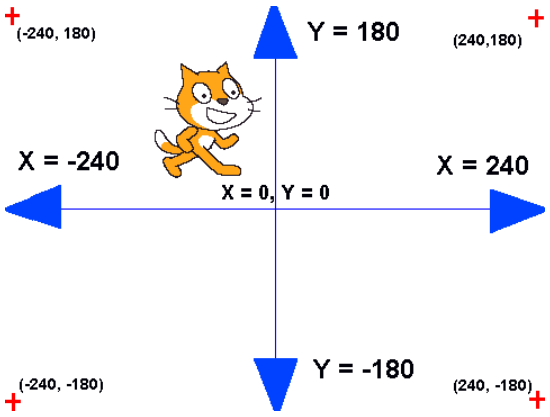
Now have the students do it on their computers.

## Coordinate System

### Explanation

Students have probably learned the x/y coordinate system in school, but here is a good opportunity to review it. Ask them what x and y stand for. Which directions are positive x and positive y?

On the whiteboard, draw a coordinate system on the board that shows the x and y axes, with (0, 0) in the center. It should look like the following diagram (although you do not need the cat or the numbers at the corners). This is the coordinate system for Scratch. You can change where a sprite is located by changing the x and y values.



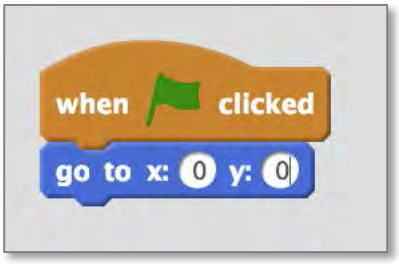

Show the students that as you move your cursor around the Scratch coordinate system, you can see the x and y values in the corner.


## Hands-on Work

Demonstrate how to move a sprite to a location. Follow these steps:

1. From the **Events** category, drag out "When green flag clicked".
2. From the **Motion** category, drag out "Go to x: __ y:___" and put it underneath.

3. Change x and y to both be zero. Your code should look like this:




4. Move the sprite to somewhere on the screen and click the green flag. It should always move to the center of the screen.

Now have the students do it on their computers. Instead of having them put it in the center, see if they can figure out the x and y values to have it be in the center of the bottom of the screen (in other words, standing on the bottom of the screen in the middle). They should have values x of zero and negative y.

## Loops

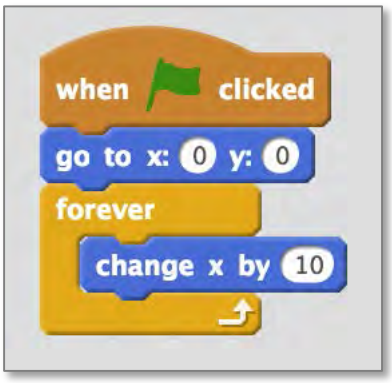
### Explanation

Explain to the students that loops repeat blocks of code over and over again. The most common loop in Scratch is the "forever" loop, which repeats forever, as long as the project is running.

### Hands-on Work

Demonstrate how to use the forever loop to move a character smoothly. Follow these steps:

1. In the sprite's code, from the **Control** category, drag a **forever** block and add it to the bottom.
2. From the **Motion** category, add a **change x by 10** block and put it inside the **forever** block. Your code should look like this:




3. Click the green flag. The sprite should move to the right until it hits the edge of the screen. Scratch will usually prevent sprites from going off the edge of the screen.

Now have the students do it on their computers. Have them play around with the value in the **change x by** block. What happens if they make it 20? -5? Advanced students can figure out how to make the sprite go up instead of across.

## Conditionals

### Explanation

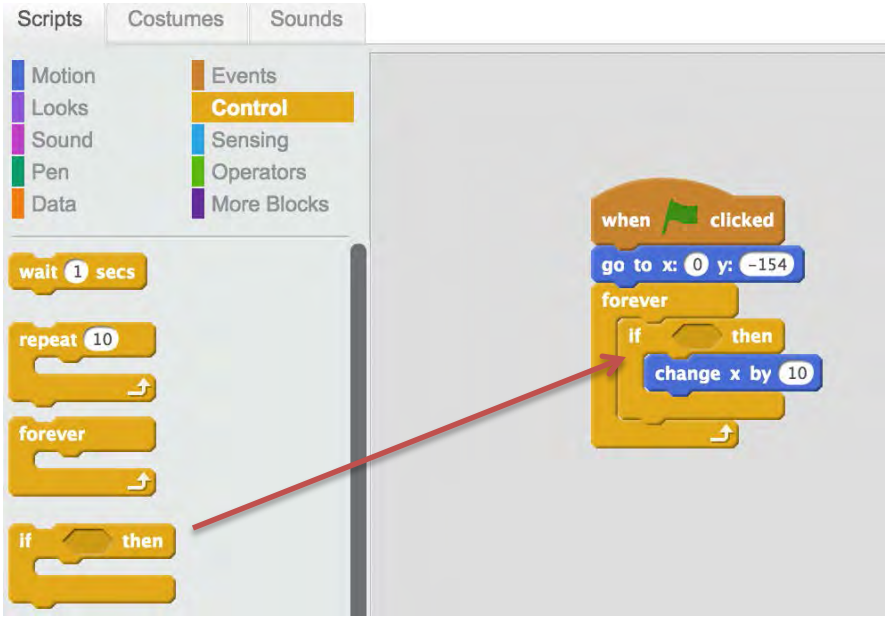
Explain to the students that you now want to make the sprite move with the arrow keys. This means you have to check *if* an arrow key is pressed, and *then* move it. An if/then statement is called a conditional. The blocks inside the if/then statement are only done if the statement is true. In other words, if the conditions are right.

You need to be checking constantly if the arrow keys are being pressed, so that means the if/then statement needs to be inside the forever loop.

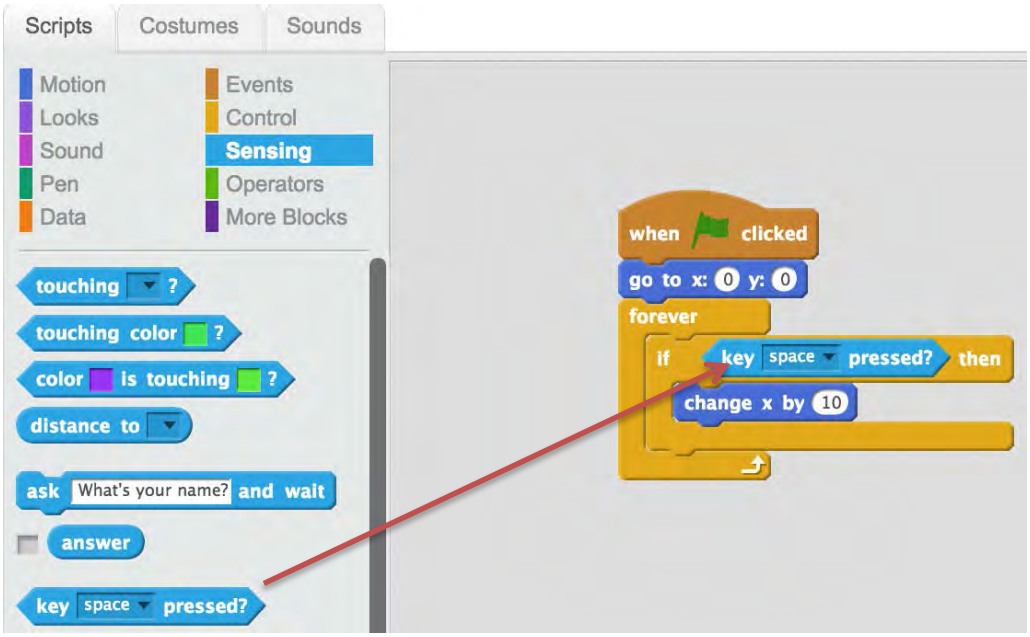
### Hands-on Work

Demonstrate how to use the if/then statement to move the sprite with arrow keys. You will show the students how to move the sprite left and right. They will then need to figure out how to move it up and down. Follow these steps:

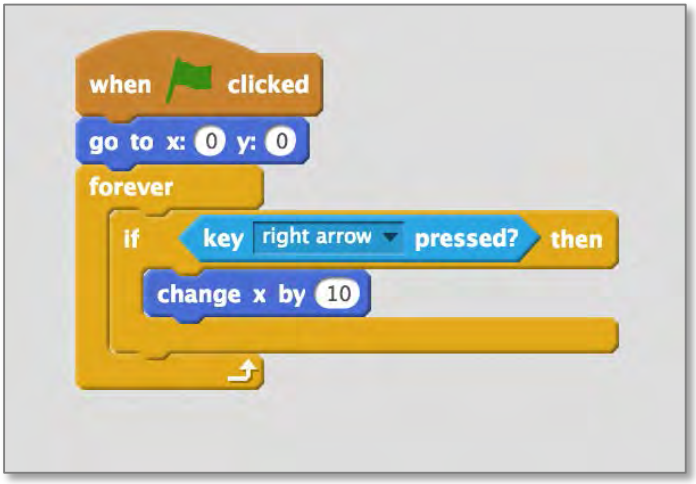
1. In the sprite's code, from the **Control** category, drag an **if/then** block and add it around the **change** x block.



2. You will notice that there is a diamond-shaped hole in the **if/then** block. To fill that hole, go to the **Sensing** category and drag an **if space key pressed** into that hole.

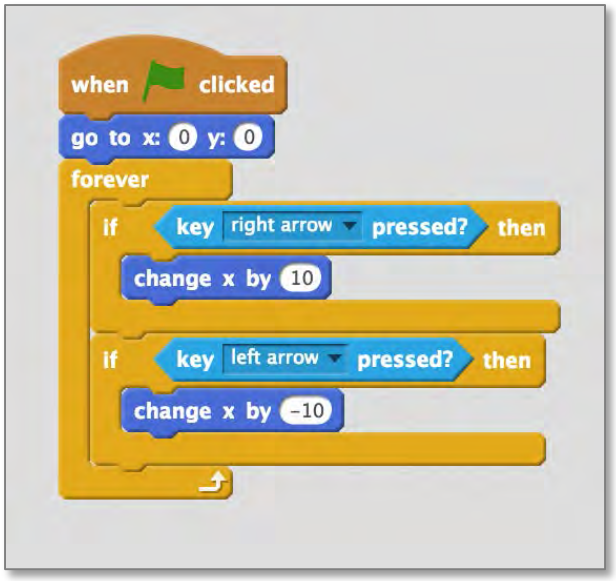



3. From the drop-down in that box inside the block you just dropped, change “space” to “right arrow. Your code should look like this:




4. Click the green flag. Now, when you press the right arrow key, the sprite will move to the right.
5. Next, add the ability to move to the left. We will just copy our code and modify it. In your code, right-click where it says “if” and choose “Duplicate.” Move the new if block under the old one. Be sure it is not inside, but below it.
6. Change the “right arrow” to “left arrow”.

7. Change 10 to -10 so that it will move to the left. Your code should look like this:

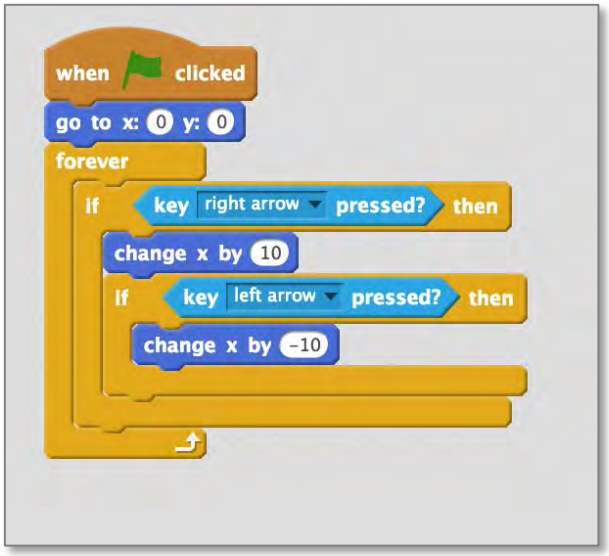



8. Click the green flag. Your sprite can now move left and right with the arrow keys.
9. One last thing: show the students how to give their projects a unique name. If you are sharing a class account, you may want them to name the project with their own names. The name field is in the top left corner.


Now have the students do it on their computers. When they have done left and right, challenge them to add up and down.

Make sure the students' games are saved in their accounts. To do this, all they need to do is sign in and give the project a unique name. Once they are signed in, the project will be automatically saved.

**Troubleshooting:** A common mistake is for the if/then statements to be inside of each other. This will make it so that more than one key needs to be pressed at the same time. Make sure that each if statement is underneath the previous one and not inside. Drag it back out and down if it is inside. Here is what it will look like if it is *incorrect*:



## Practice Activity

At the end of class, ask students:

- How do you create a new sprite?
- In an x/y coordinate system, which way is x and which way is y?
- In Scratch, what block do you use to make blocks repeat?
- What block do you use if you only want blocks to run if something is true?
- What blocks do you need to move a sprite around with the arrow keys?

## Rubric

For the Scratch move the sprite project, use the Rubistar Storyboard - multimedia rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=2&section_id=3&

Use the rubrics: Content and Required Elements

# Lesson 5: Variables and Random Numbers

Middle school: Session 5
High school: Session 4

## Summary

Students learn about variables and random numbers in Scratch. Variables are a method to store, retrieve, and display data. Students will build on their project from last time to add a sprite that moves randomly and a score using variables.

**Note:** If a student missed last time, they can "remix" another students' project to be able to continue it. Have the student who is up to date share their completed project, and then have the student without the project log into Scratch, go to the URL of the shared project, and then click **Remix**.

## Learning Objectives

After doing this session, students should be able to:

- Understand what a variable is
- Understand what random means
- Create and use a variable in Scratch
- Create and use a random number in Scratch
- Create a simple game where a sprite tries to touch another sprite

## Materials

- A computer for each student with internet
- Teacher's computer with internet
- Projector
- Whiteboard or equivalent
- Projects from last session

## Preparation

Read through the **Discussion** section so that you make sure you understand it and go through the steps in the **Lesson Procedure** section and make sure you can do them.

## Vocabulary

The following vocabulary is used in this lesson:

- **Variable** – In computer programming, a variable is a place in memory where data can be stored, modified, and retrieved.
- **Random** – Unable to be predicted.

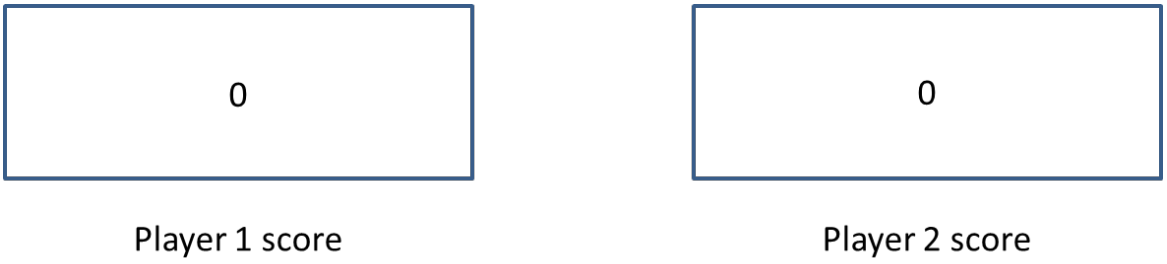
## Introduction

In this session, you will lead students through creating a simple game where one sprite tries to catch another sprite. There are two concepts you will cover: variables and random numbers. You will first have a discussion about these concepts. You will demonstrate how to create the game, and then have the students do it themselves. While you are demonstrating what to do, make sure the students have their hands off the computer.

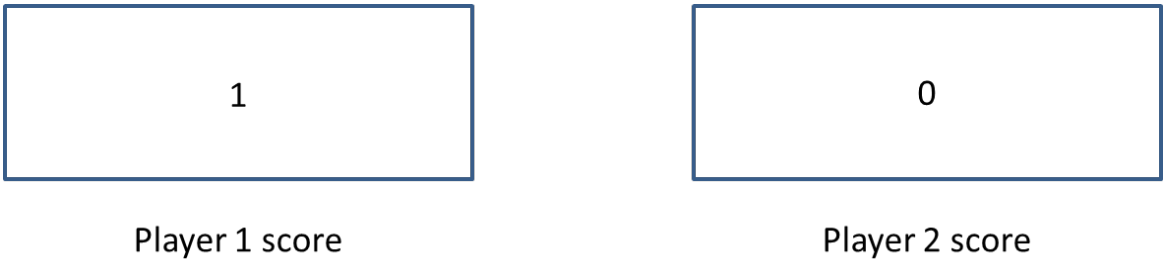
## Discussion

Explain to the students what a variable is. If they have learned algebra, they have heard of variables, but software variables are a little different. They are a place in the computers' memory where information can be stored and changed. Ask the class for ideas of where variables might be used in video games – in other words, when do video games store information? Examples include the score, health points, character names, etc.

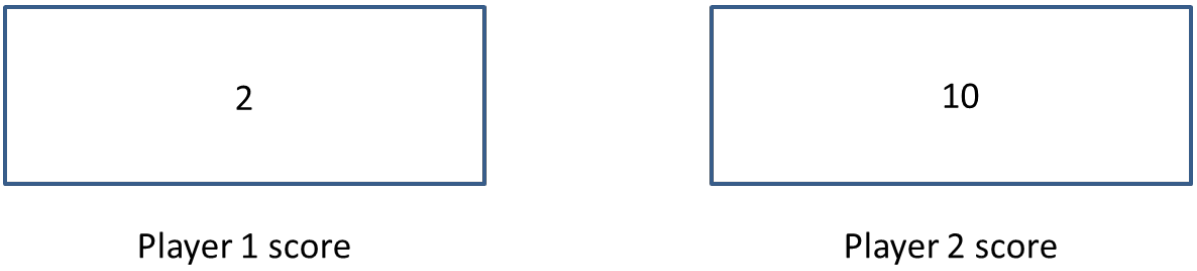
To give a more concrete example, draw two boxes on the white board. Under one, write "Player 1 score". Under the other, write "Player 2 score". These represent two variables in a two-player game. When the game starts, ask them what values should be in those boxes. The answer is zero for both. Write 0 in both boxes.




Now, say that Player 1 makes a point. What should be in the Player 1 score now? The answer is one. So erase the 0 and replace it with 1.



Let's say Player 1 makes another point. What should be Player 1 score now? Again, replace it with 2. Each time, you are changing it by 1. What if Player 2 did something special and got 10 points? Where would we put that value? Change Player 2 score to 10.




This shows you how you can set and change variable values.

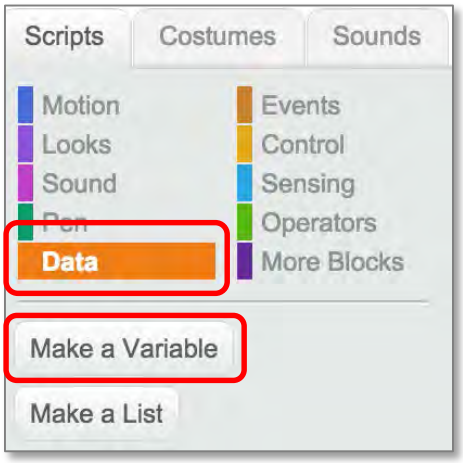
Next, talk about the concept of random. Ask the students what random means. Often students think about random in terms of phrases like "That's so random". Guide them towards thinking of phrases like, "Pick a random number between 1 and 10." See if they can figure out that random means unpredictable. In the case of the random number, it is a number that cannot be predicted.

Ask them for ideas about where random numbers might be used in a game. Often the places and times where opponents and targets appear in a game are random.

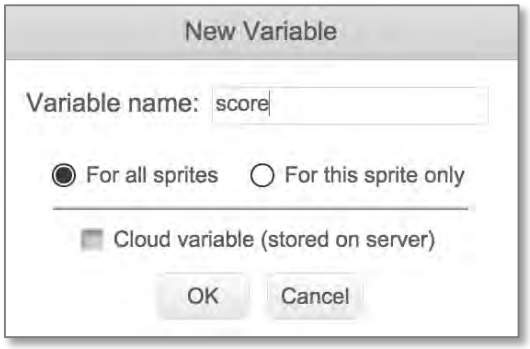
## Lesson Procedure

Demonstrate how to add a variable and use random numbers by adding to the project that you created last time. Follow these steps:

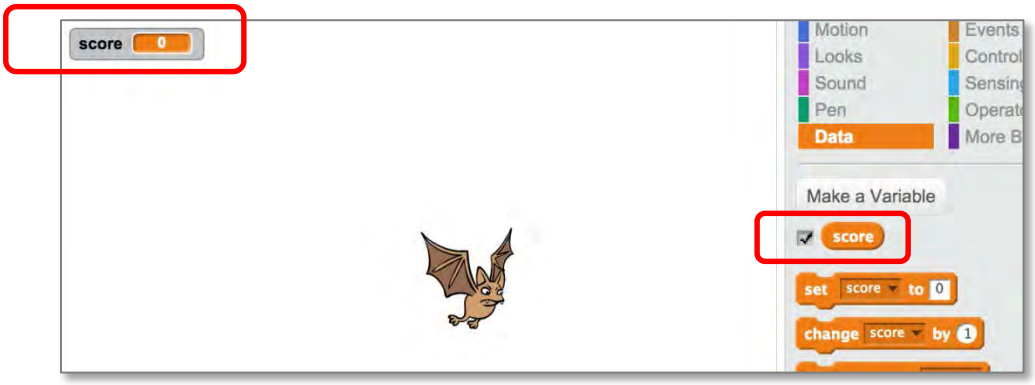
1. Sign into Scratch.
2. Click on your username in the top right corner and choose **My Stuff**.
3. Open your project from last time.
4. First, you will add a variable to keep track of the score. Click on the **Data** category and then click on **Make a Variable**.



5. Name the variable “score”. Leave it as **For all sprites**. (You can have variables for just a sprite, but then they cannot be accessed by other sprites. For almost all variables, you will choose **For all sprites**.) Leave **Cloud variable** unchecked. Click **OK**.

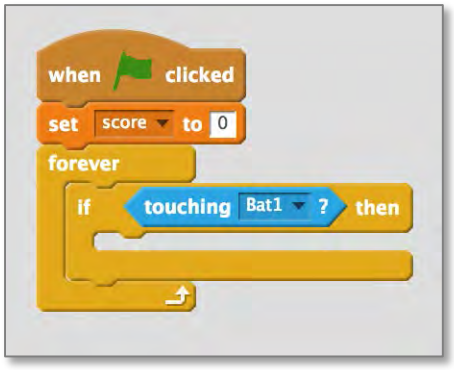



6. Note that the score has appeared on the screen. If you want to not have it visible, you can uncheck the box by the variable name. You want the score to be visible, so leave it checked.

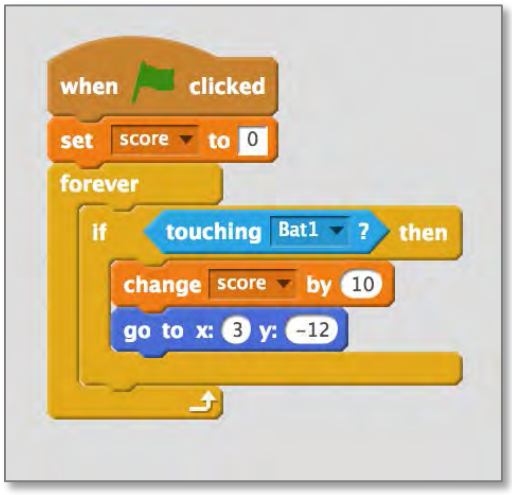



7. Let us add a target sprite. From the **New sprite** area, choose library.
8. Choose an object to be your sprite, such as the Bananas sprite. Click **OK**.
9. Shrink the new sprite so it is about the same size as the first sprite.
10. Now you will work on code for the Bananas sprite. From the **Events** category, drag the “when green flag clicked” into the code space.
11. Return to the **Data** category and drag “set score to zero” to be under “when green flag clicked”. This means that every time the game starts, the score will be set to zero. Your code should look like this:

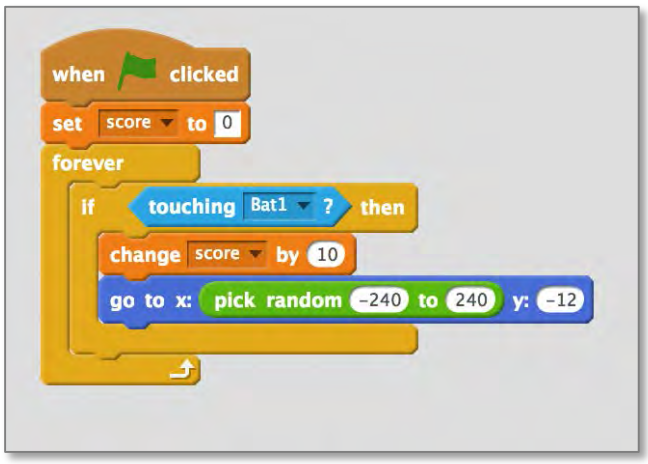
12. Now you will add code so that every time the banana touches the other sprite, it should add 10 points. We need to check constantly to see if they are touching, so under this add a “forever” block from the **Control** category.
13. Inside the “forever” block, add an “if/then” block.
14. From the **Sensing** category, drag in the first “touching” diamond so that it fits after the “if”.
15. In the drop down of the “touching” diamond, select the name of the first sprite. If you have chosen Bat1, then your code should look like this:



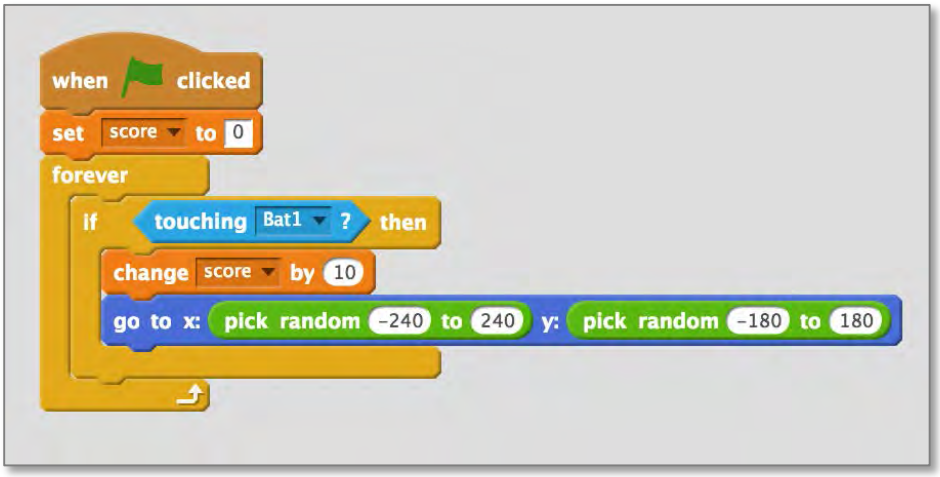

16. You need to do two things when the two sprites are touching. The first is to increase the score by 10 points. From the **Data** category, drag in “change score by 1”. Change the 1 to a 10.
17. The second thing is to move the sprite to a random part of the screen so that it can be caught again. From the **Motion** category, drag “go to x:__ y:__” under the “change score by 10”. Your code should look like this: (you may have different values for x and y).



18. We need random numbers for both x and y so that the sprite appears in a totally random place on the screen. From the **Operators** category, drag "pick random from 1 to 10" into the x: space. What is the lowest and highest x value that you want? You can put your cursor all the way to the left of the screen and read off the x value just below the screen. Then do the same for all the way to the right. The values should be -240 to 240. Put these values into the "random" block, so your code looks like this:




19. Now do the same for the y value. Add a "random block". What is the highest and lowest y value that you want? Your code should look like this:




20. Click the green flag. Use the arrow keys to move around and catch the second sprite. Each time you catch it, it moves to a random place and your score goes up by 10.

Now have the students do it on their computers. Make sure they all start with a project where the sprite can move up, down, left, and right. If not, quickly show them how to do it, or have them remix another students' project (as described at the beginning of this lesson).

Any advanced students who finish early can make a two-player game where there are two sprites that race to be able to catch the target sprite first.

## Practice Activity

At the end of class, ask students to explain:

- What is a variable?
- What does random mean?
- How do you create a variable in Scratch?
- How do you create a random number?
- Check that everyone was able to create a game where one sprite tried to touch another?

## Rubric

For the Scratch game, use the Rubistar Storyboard - multimedia rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=2&section_id=3&

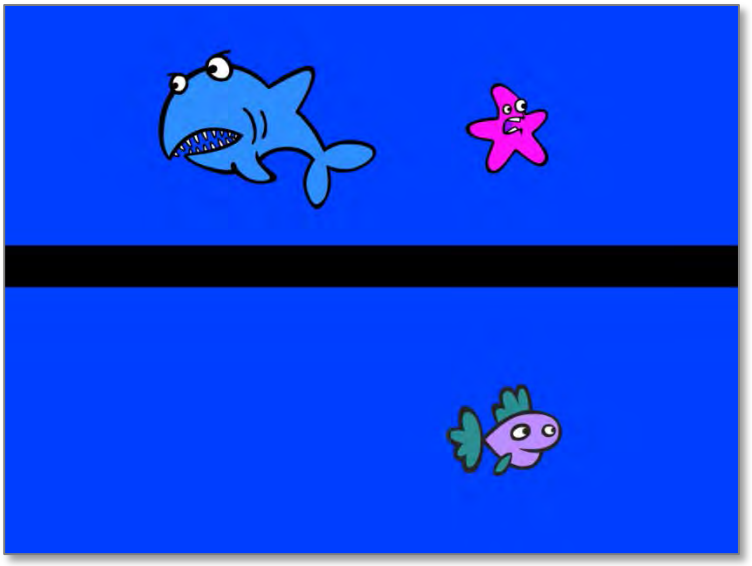
Use the rubrics: Content and Required Elements

# Lesson 6: Aquarium Simulation

Middle school: Session 6
High school: Skip this lesson

## Summary

Students learn about how to make a simple aquarium simulation to help someone make a decision about which fish should be placed together in one tank.



## Learning Objectives

After doing this session, students should be able to:

- Understand how to model a simple system
- Create a simple simulation with three sea creatures
- Change the direction a sprite is pointing

## Materials

- A computer for each student with internet
- Teacher's computer with internet
- Projector
- Whiteboard or equivalent

## Preparation

Read through the **Discussion** section so that you make sure you understand it and go through the steps in the **Lesson Procedure** section and make sure you can do them. You can see a version of the project here:

Aquarium: https://scratch.mit.edu/projects/68659588/

## Vocabulary

The following vocabulary is used in this lesson:

- **Direction** – In Scratch, a value from 0 degrees to 359 degrees that indicates in which direction a sprite is pointing, and also the direction it will move with a "move" block.

## Introduction

In this session, you talk about how simulations can be used to test out scenarios before making a final decision. This session is an example of how to create a simple simulation. The model will contain rules on how animals move and what they eat. You will show them how to start an aquarium project with two animals, and they will repeat what you have done and add a third animal.

## Discussion

Start by talking about this situation: you are someone who owns two aquariums, and you have three animals to put in them: a starfish, a fish, and a shark. You want to figure out how to put them in tanks so that they do not eat each other. You are going to use a simulation so that you do not accidentally lose any animals in trying to figure this out.

Recall that simulations have models and visualizations. Let us see if we can create a simulation with a model for these three animals. The visualization will look like an aquarium with the animals moving around.

One more important fact about Scratch: sprites have a direction. You can point a sprite in any direction, and then when you tell it to "move", it will move in that direction. The direction is an angle in degrees that ranges from 0 (up) to 360 (up again), where 90 is pointing right. Sprites also have a rotation mode, which can be one of three modes: (1) rotate to point in their direction, (2) only flip left and right, and (3) never flip or rotate at all. By default, the sprites will rotate to point in their direction.

## Lesson Procedure

Start by writing the model on the white board. First, there will be three rules for how fast the animals move:

1. The starfish moves slowly
2. The fish moves at a medium speed
3. The shark moves quickly

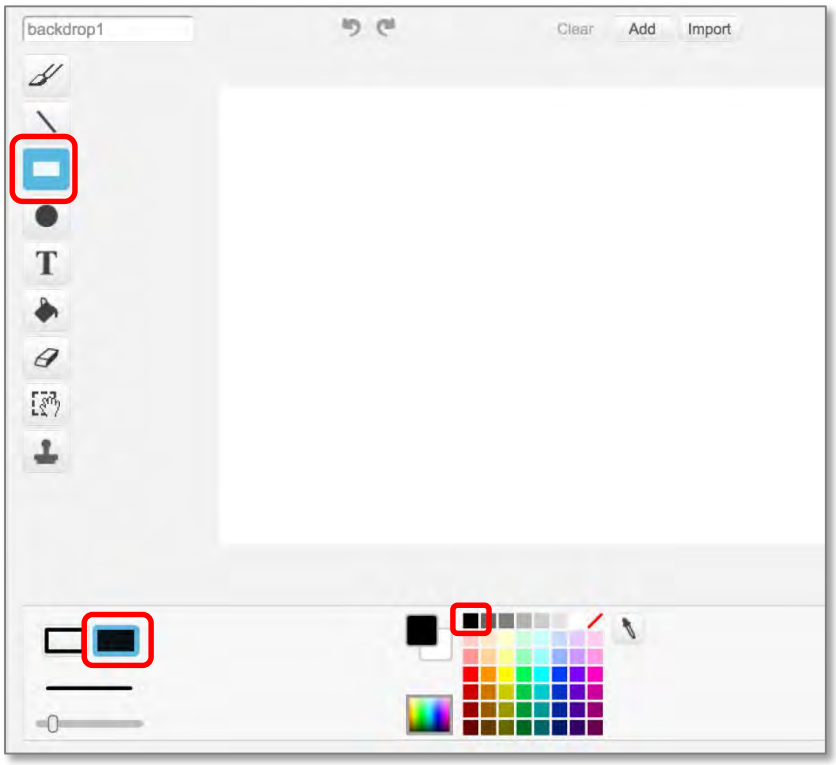
Next, add two more rules about who eats whom:

4. The fish eats starfishes
5. The shark eats fish

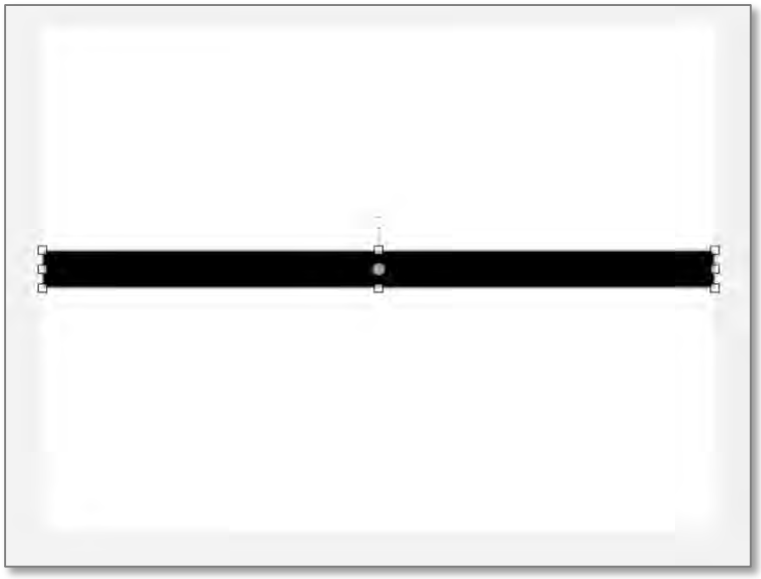
Start a new Scratch project to make the visualization:

1. Delete the cat.
2. At this point, you only have the Stage in the sprites area. Click on the tab **Backdrops**, which is contains the graphics for the background of the game.

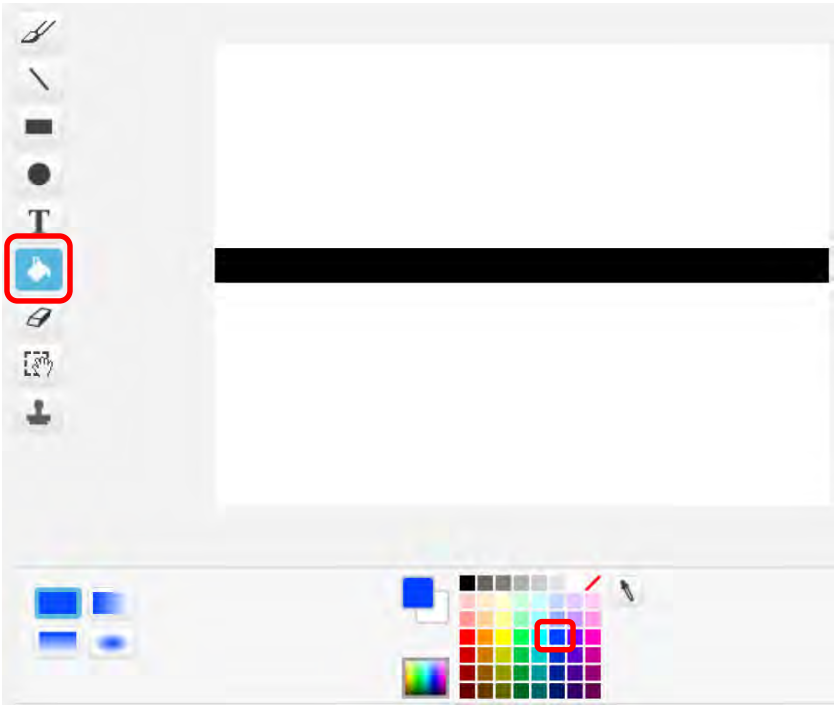

3. Make a divider line between the top and bottom aquarium. Click on the rectangle tool, then the solid color, and then black.



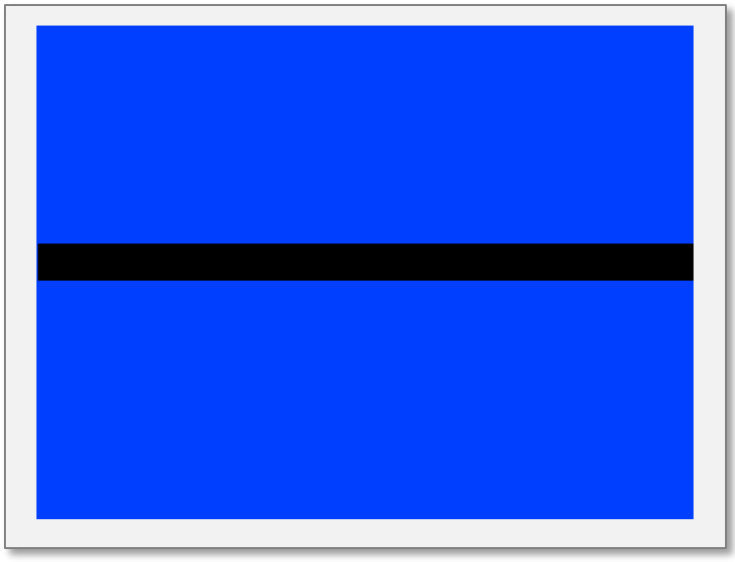
4. Draw a line horizontally in the middle of the white rectangle.



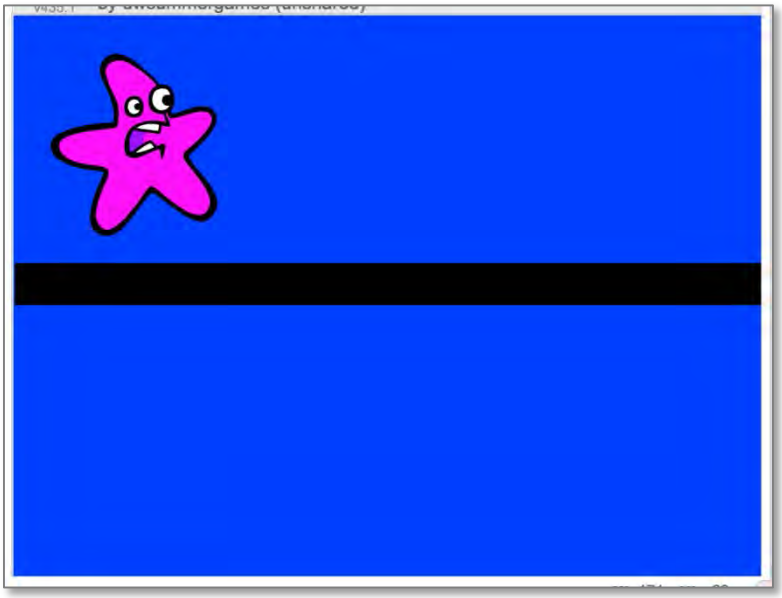
5. Click on the paint bucket (fill) and blue.



6. Click above and below the black line to fill the rest of the screen with blue.

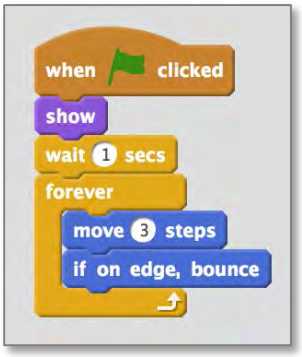


7. Now we have our two aquariums, one on top of the other. Let us add the first animal. Add a new sprite from the library. Click on the category **Animals**, and scroll to the bottom. Choose the Starfish sprite.
8. Drag the sprite into the top aquarium.



9. Now, let us add code. From the **Events** category, drag "when green flag clicked" into the code section.
10. First, let us make sure the sprite is always visible when we start. From the **Looks** category, drag "show".

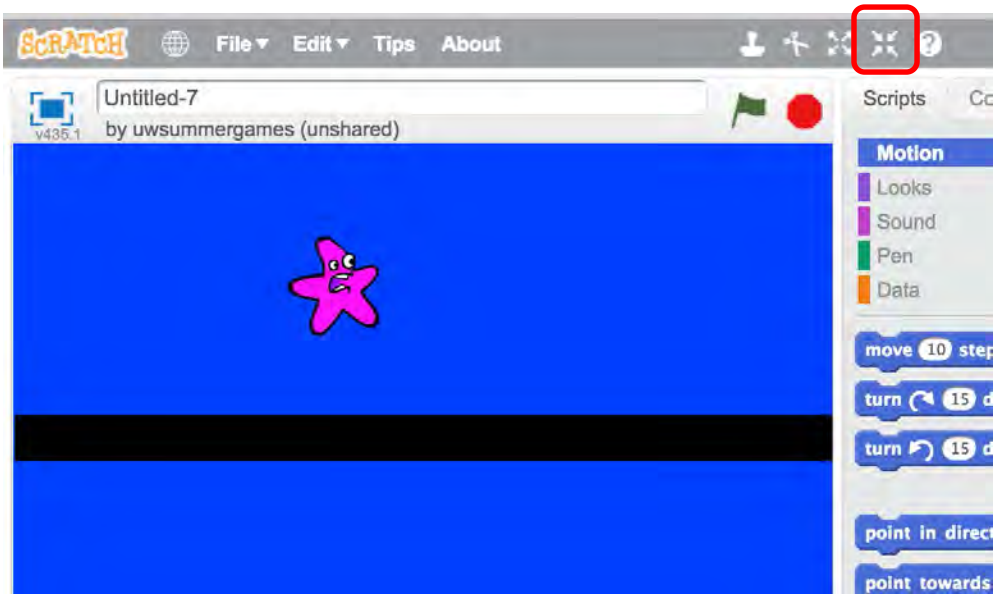
11. Next from the **Control** category, add “wait 1 sec”. This will give people the time to move the animals around before they start to eat each other.
12. From the **Control** category, drag “forever” to the bottom of the script.
13. From the **Motion** category, drag “move 10 steps” to inside the forever loop.
14. From the **Motion** category, drag “if on edge, bounce” into the forever loop. This means that when it touches the edge of the screen, it will change the direction to point it back the other way. Your script should look like this:



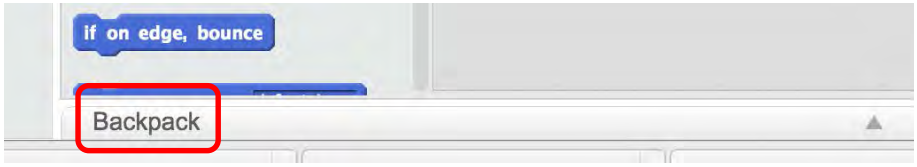

15. Click the green flag. The starfish will move back and forth across the aquarium. Ask the students if they notice anything odd. The starfish flips upside down! That is because it is rotating with the direction.
16. Click on the button near the sprite in the Sprites section to open up the Sprite information.


17. Click the rotation style with the double arrow. The starfish should stop flipping upside down and now just flip left and right.

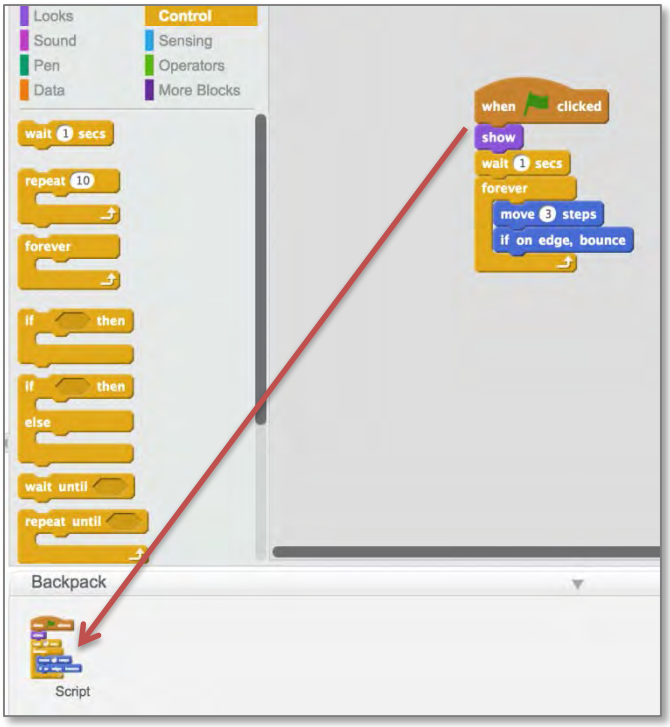
18. Click the triangle in the blue circle to close the sprite information.
19. Point back to your model. The starfish should be moving slowly. Ask the students how to do this. The answer is to change “move 10 steps” to a lower number, like “move 3 steps”.
20. You should also shrink down the sprite to make it more realistic. Click on the shrink tool at the top, and then click several times on the sprite to make it about half the size.



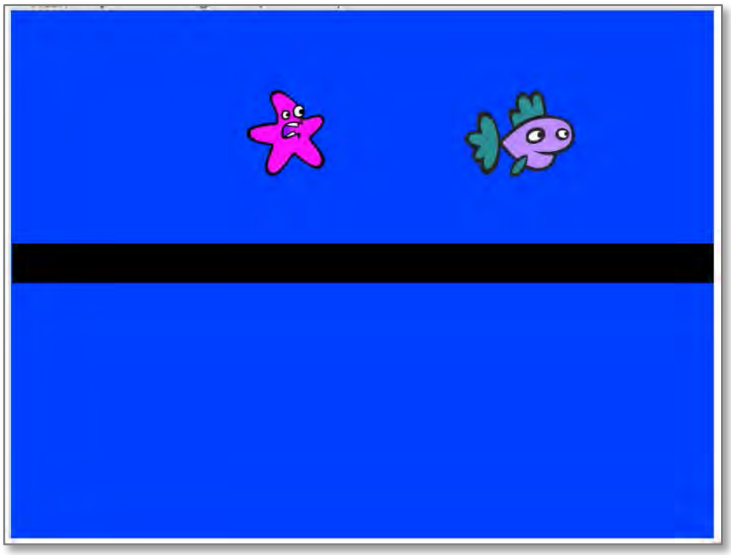

21. This code that we created will be useful for our other animals. There is an easy way to save code for reuse. Click on the Backpack at the bottom of the screen.



22. Drag the code into the backpack.



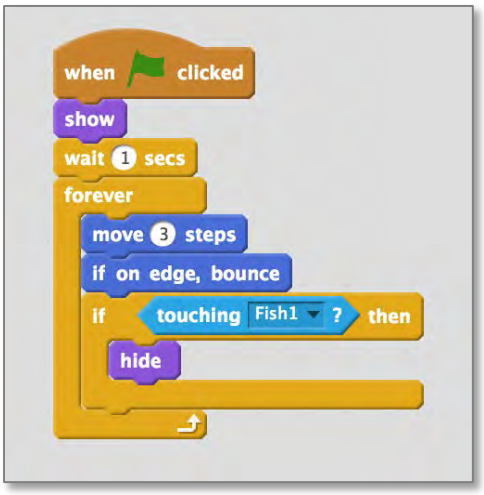

23. Click on the Backpack triangle to close it.
24. Let us add the fish. Click on the New Sprite icon and choose Fish1.
25. Drag it into the upper aquarium and shrink it a little.



26. Open up the Backpack and drag in the Script we just put there. Click the green flag and both animals now move back and forth. You will also have to change the rotation style of the fish so it does not flip upside down.

27. Point the students to look at the model. What needs to change?
    1. The fish should move faster.
    2. The fish should eat the starfish if they touch.

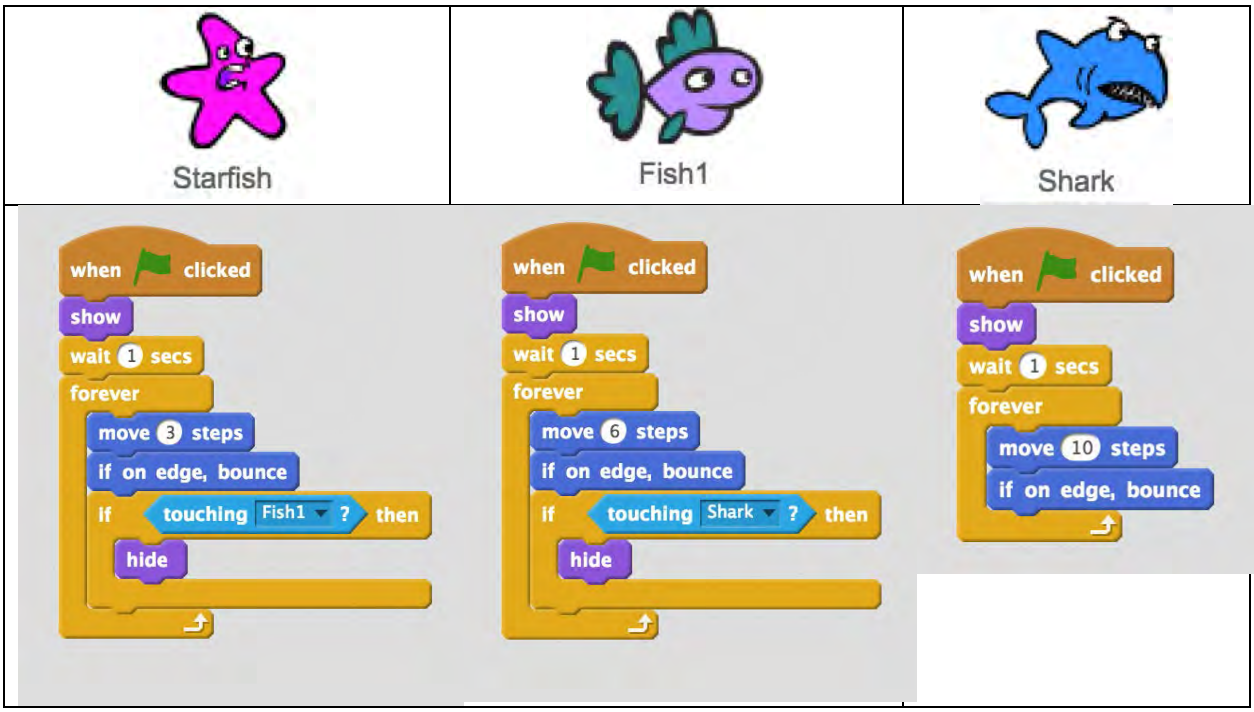
28. The first one is easily fixed. Change “move 3 steps” to “move 6 steps” in the fish’s script.
29. The second one requires some new blocks. Return to the Starfish sprite and add to its script some blocks that will check if it is touching the fish. (**Important:** Do not forget to move back to the Starfish sprite so that you are working with its scripts.)
30. From the **Control** category, drag an “if/then” block into the forever block, under the “if on edge, bounce” block.
31. From the **Sensing** category, drag a “touching” diamond into the space next to “if”.
32. On the dropdown in the “touching” diamond, choose **Fish1**.
33. From the **Looks** category, drag “hide” to **i**nside the “if/then” block. Now you have code that checks if the starfish is touching the fish, and if it is, then it will disappear. Your code should look like this:




34. Click the green flag. Now, when the fish touches the starfish, it disappears.

## Project

Have the students recreate the aquarium that you have created so far. Then have them add the shark. They will need to add code to the fish's script so that when it touches the shark, it disappears. Here is what the final scripts will look like:




Once students they have this, see if they can arrange the three animals in the two aquariums so that nobody is eaten.

Advanced students: Have them modify the model so that there are two fish in addition to the shark and the starfish.

1. Fish 1 eats starfish.
2. Fish 2 eats fish 1.
3. Sharks eat fish 2.

## Practice Activity

At the end of class, ask students:

- What is the model of the aquarium simulation?
- What is the visualization?
- How to you change the direction a sprite is pointing?
- Did everyone successfully create the aquarium model?

## Rubric

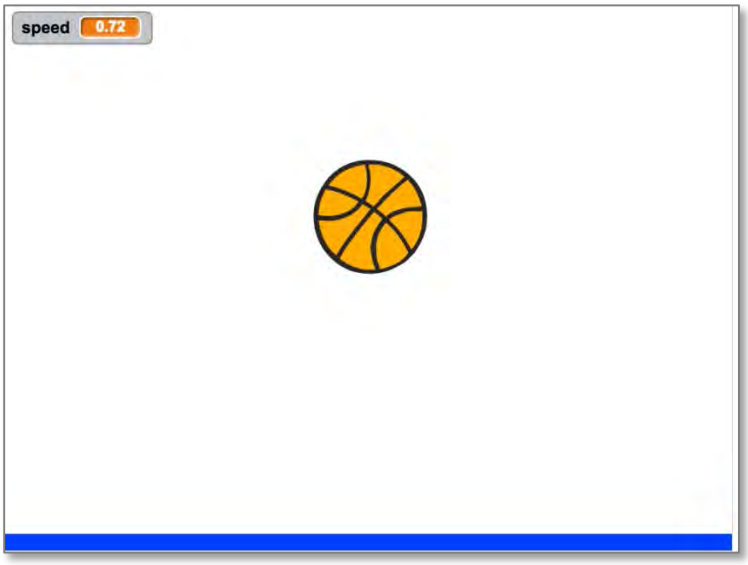
For the simulation, use the Rubistar Website Design rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=29&section_id=3
Use the rubrics: Graphics, Layout, Content Accuracy (is it a good simulation?), and Learning of the Material.

# Lesson 7: Gravity Simulation

Middle school: Session 7
High school: Session 5

## Summary

Students learn about the physics concepts of speed and acceleration, and that gravity accelerates objects downwards. Students build a project in Scratch that illustrates these concepts. Then they build their own project that has a jumping sprite.




## Learning Objectives

After doing this session, students should be able to:

- Understand how gravity is modeled
- Create a simple simulation where a sprite drops through gravity

## Materials

- A computer for each student with internet
- Teacher's computer with internet
- Projector
- Whiteboard or equivalent
- A small object that you can drop

## Preparation

Read through the **Discussion** section so that you make sure you understand it and go through the steps in the **Lesson Procedure** section and make sure you can do them. You can see versions of the projects here:

Falling ball: https://scratch.mit.edu/projects/68659768/

Jumping sprite: https://scratch.mit.edu/projects/68659884/

## Vocabulary

The following vocabulary is used in this lesson:

- **Acceleration** – The rate at which velocity (speed) changes.

## Introduction

In this session, you talk about the physics of gravity, how it is an acceleration, and what acceleration means. This session is an example of how to create a simple physics simulation. The model describes what speed a sprite will move at, and then the visualization is the falling sprite. You will show them how to create a project with a falling sprite. Then, they will build a project with a jumping sprite.

## Discussion

Start by asking students about gravity. This is something they experience all of the time, and yet have probably not thought much about. Ask: what is gravity? How does it work? Do heavier objects fall faster than lighter objects? (The answer to that last one is "no," except in the case where lighter objects are slowed down by air resistance.)

No one knows exactly how gravity works, but physicists have figured out that gravity is an acceleration. Acceleration is when the speed of something changes. The gas pedal of a car is called an "accelerator", because when you push it down, the speed of the car gets larger.

Think about this: when the driver pushes down the gas pedal, the car accelerates, and you are pushed back into the seat. If you were sitting in a car that was parked, facing up a very steep hill, you would also be pushed back into the seat. It is the same effect. This is because gravity is an acceleration.

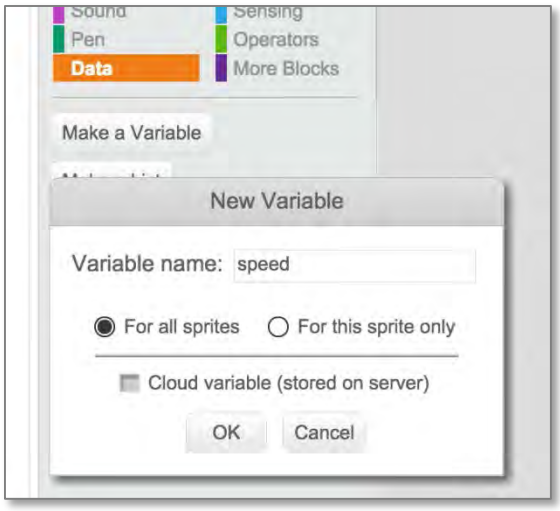
Recall that simulations have models and visualizations. Let us see if we can create a simulation with a model for gravity. The visualization will be a falling sprite.

## Lesson Procedure

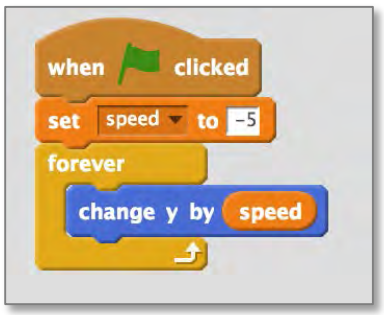
Follow these steps to create a new project with a gravity model. Do not have the students follow along.

1. Create a new Scratch project.
2. Delete the cat.
3. Add a new sprite from the library. Choose the basketball.

4. Create a variable called “speed”. From the **Data** category, click **Make a Variable** and name it speed.




5. From the **Events** category, drag “when green flag clicked” into the code section.
6. From the **Data** category, drag “set speed to 0” under “when green flag clicked”. Change the 0 to -5 so that the speed will be downwards.
7. From the **Control** category, drag “forever” to the bottom of the script.
8. From the **Motion** category, drag “change y by 10” to inside the forever loop.
9. From the **Data** category, drag “speed” into where it says “10” so that is now says “change y by speed”. Your script should look like this:



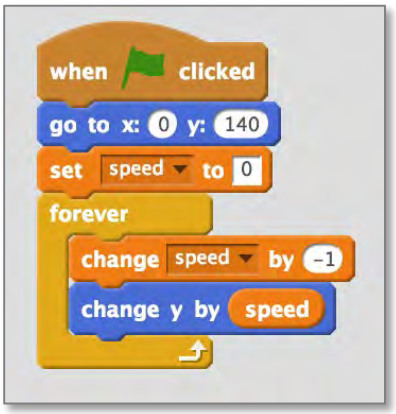

10. We have just created a simple model that has the ball fall at a constant speed. Click the green flag to try it out. Ask the students: does this look like how a ball falls? It really does not.
11. Stop the program ( ), drag the ball to the top again and try changing the -5 to other values: -10, -1, 3, etc. Do any of those look right?
12. Take a small object (like a ball or a pen) and drop it and have the students observe it. Does it always move at the same speed? Ask for a volunteer and have them catch the object one inch below where you drop it. Then have them catch it near the floor. Ask them for which one was it faster? They should be able to notice that the object moves slowly at first and then faster. We

need a model where the speed is constantly getting bigger, but in a negative direction, which is downward for y.

13. On the whiteboard, write these rules for gravity as an acceleration:

> **Gravity Model**
>
> 1. When an object is first dropped, its speed is zero.
> 2. Each time in the forever loop, the speed becomes a little more negative
> 3. Each time in the forever loop, the y value changes by the speed

14. Stop the project if it is running.
15. Add code to always start in the same place, at the top of the screen. From the **Motion** category, drag a "move to x:_ y:_' block under the "when green flag clicked". Change the x and y values so it reads "move to x:0 y:140"
16. Change the "set speed to -5" to "set speed to 0". This is the first rule in the model.
17. Add a new line in the forever loop that changes speed by -1. (From the **Data** category, choose "change speed by 1" and change the 1 to -1.) This is the second rule in the model. The third rule in the model is already there (change y by speed). Your code should look like this:



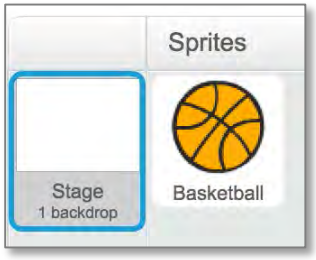

18. Click the green flag. It looks more realistic now, doesn't it?
19. Stop the project, change the -1 to -0.1, and click the green flag. What does that do to the gravity? What if it's -10? What if it's 0.1?

## Project

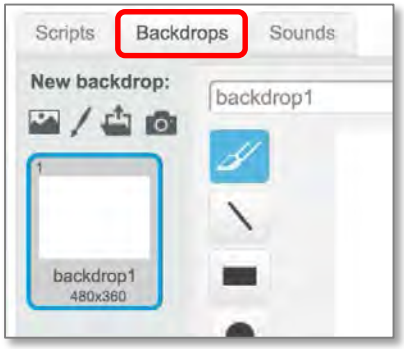
Students will add to the gravity model to make a jumping character. First, have them choose a character and make it fall, just like you did in the demonstration.

Once students have this, then they need to add a floor. Show them these steps to do that:

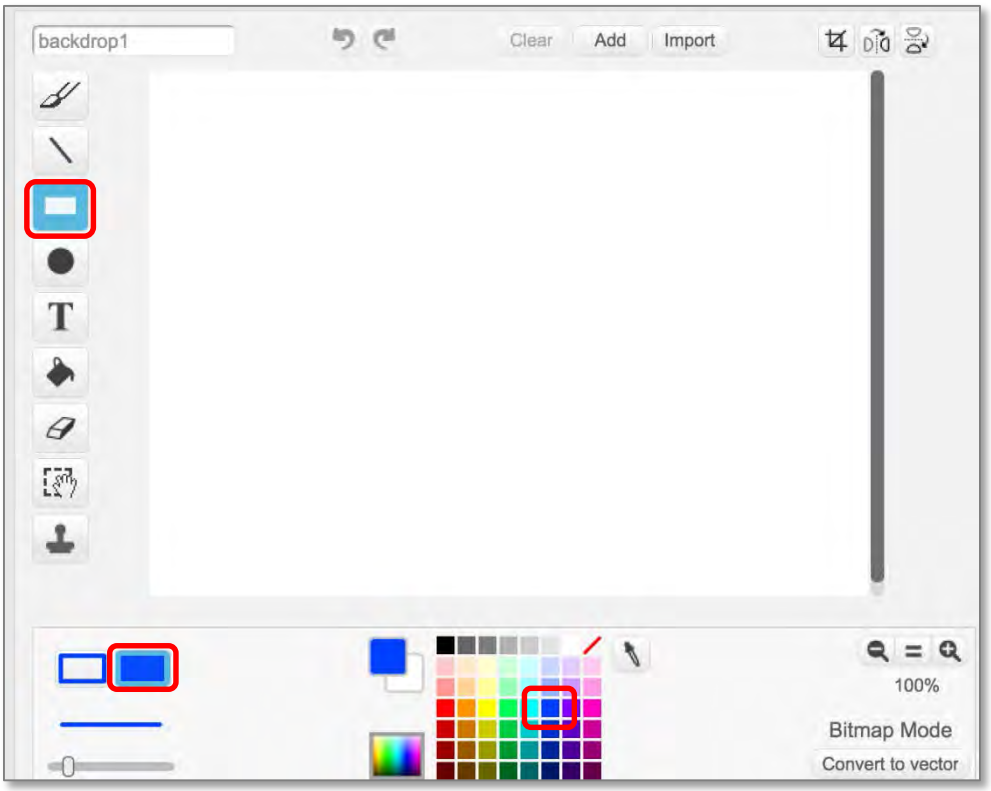
1. In the **Sprites** section, click on the **Stage**.



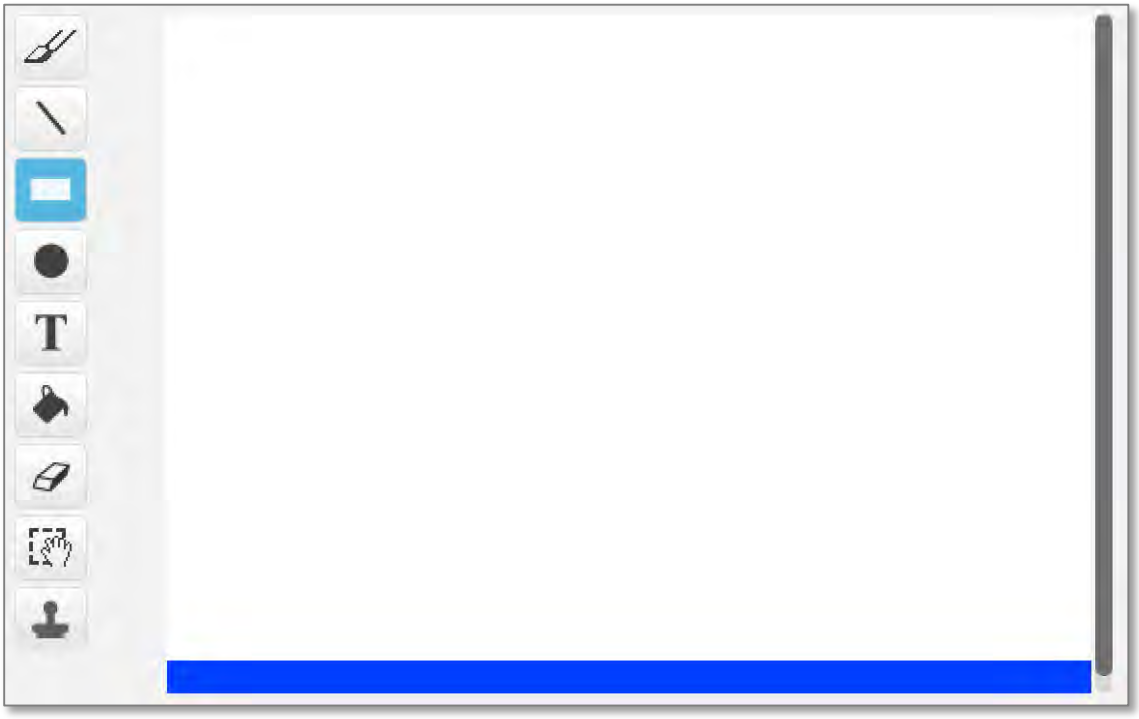

2. Click on the **Backdrops** tab.




3. Click on the rectangle tool and then the solid rectangle and the color blue. (Or whatever color they want.)



4. Drag a thin rectangle across the bottom to create a blue floor.



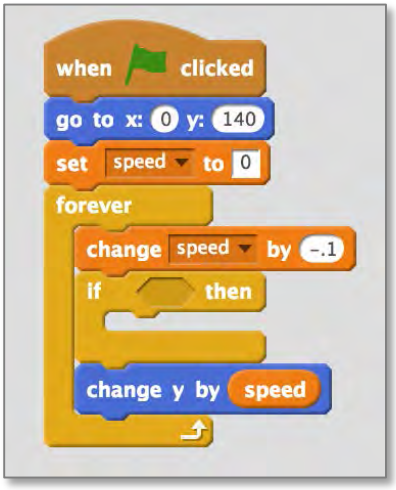
Given them a couple of minutes to add the floor, and then go back to the whiteboard. On the whiteboard, add these new rules to the model for jumping:

| |
|---|
| 4. Sprites do not fall if they are touching the floor. (Speed is zero.) |
| 5. A sprite can only jump when it is touching the floor. |
| 6. To jump, set the speed to a positive number. (Moving upwards.) |

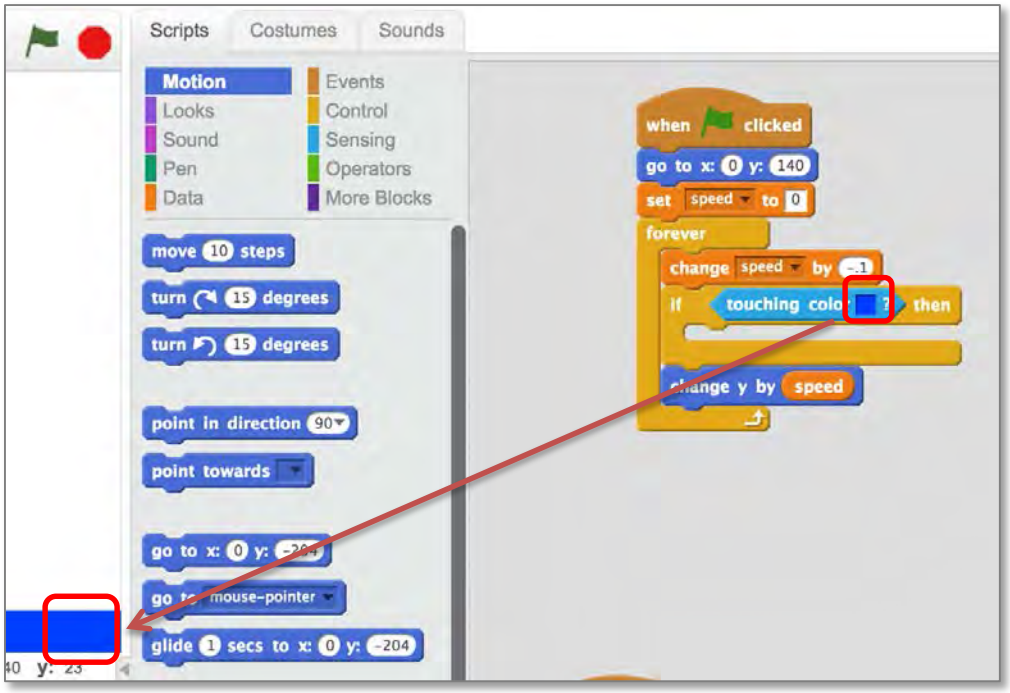
In this case, we want to check if the sprite is touching the floor. If it is, set the speed to zero.

1. Click on the basketball and then the Scripts tab.
2. Set the “change speed by” to be -0.1, so that it will fall slowly.

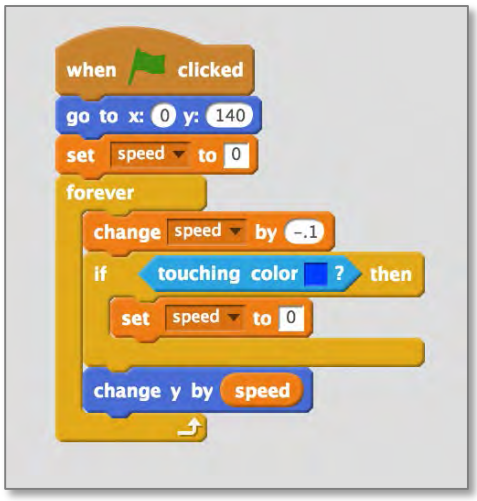
3. From the **Control** category, drag an "if/then" block into the forever loop before "change y by speed". Your code should look like this:




4. Now to check if it has touched the floor. We will do that by checking if it is touching the floor's color. From the **Sensing** category, drag "touching color?" into the empty diamond after the "if".
5. Next, we need to select the color. Click on the colored box and then floor. This will make the colored block turned blue.



6. Finally, from the **Data** category, drag "set speed to 0" into the if/then block. Your final code should look like this:

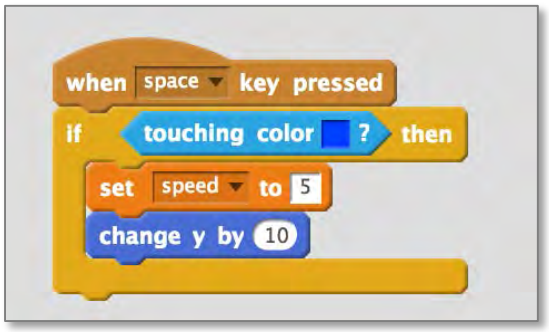



7. Click the green flag. Note that the ball now stops on the floor.

Have the students add this code so that their sprite falls to the floor. Next, ask them to add code so that the sprite jumps when the space bar is pressed. Give them these hints:

1. In the **Events** category, there is something called "when key space pressed". Drag this to a new part of the screen. This code will be called when the space bar is pressed.
2. Check to see if the sprite is touching the floor. It's the same way you showed them, with color.
3. If it is, then set speed to 5.
4. Move the sprite a little up so that it is not touching the floor.

Their added code should look like this:




Pressing the green flag and then the space bar should make the sprite jump.

Advanced students: Add in the ability to move the sprite left and right, as they did in Session 4. Go back to the background and add some other horizontal blue lines to make a platform game. Anything that is blue will act as the floor and stop the sprite from falling. (See picture below.)


## Practice Activity Check

At the end of class, ask students:

- What did you learn about how to simulate gravity?
- How was a variable used?
- Did everyone successfully create a jumping game?

## Rubric

For the simulation, use the Rubistar Website Design rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=29&section_id=3

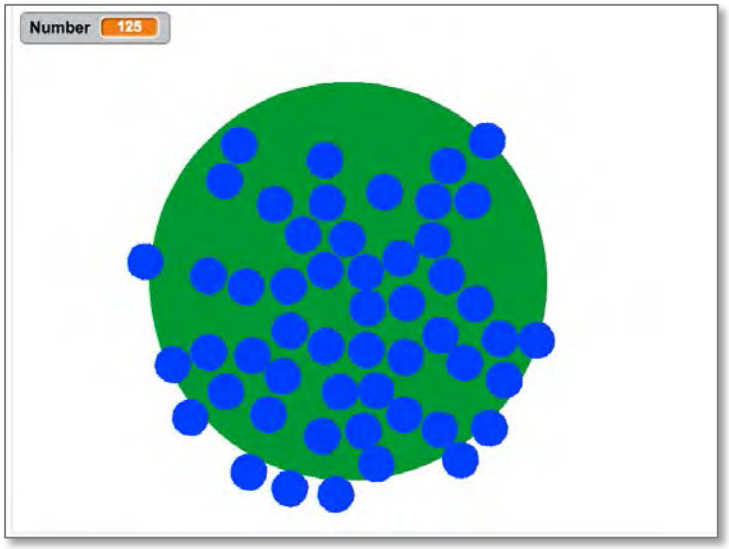
Use the rubrics: Graphics, Layout, Content Accuracy (is it a good simulation?), and Learning of the Material.

# Lesson 8: Bacteria Simulation 1

Middle school: Skip this lesson
High school: Session 6

## Summary

Students learn how bacteria multiply exponentially under favorable conditions. Then they learn about how cloning works in Scratch to create copies of existing sprites.




## Learning Objectives

After doing this session, students should be able to:

- Understand how bacteria multiply under favorable conditions
- Clone sprites in Scratch to create copies of an existing sprite

## Materials

- A computer for each student with internet
- Teacher's computer with internet
- Projector
- Whiteboard or equivalent

## Preparation

Read through the **Explanation** sections so that you make sure you understand them and go through the steps in the **Lesson Procedure** section and make sure you can do them. Also, try doing the **Cloning Project** if you have time. You can see a version of the project here:

Throwing basketball: https://scratch.mit.edu/projects/68659942/

## Vocabulary

The following vocabulary is used in this lesson:

- **Clone** – In Scratch, duplication of a sprite, including its code, position, direction, and variables.

## Introduction

In this session, you talk about how bacteria grow in favorable conditions. Students will learn the Scratch concept of cloning, which allows them to make multiple copies of existing sprites. Then they make a simple project that shows how you can use cloning to create a projectile, in this case, a thrown basketball. They will create a bacteria simulation next session.

## Bacteria Explanation

Explain that in this session, you will talk about simulation of biological systems, in particular, bacteria. Bacteria are single-cell, microscopically small organisms. Ask students what they know about bacteria. They can create diseases, break down dead plants and animals, create yogurt, etc.

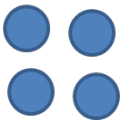
On the whiteboard, draw a circle and say that is one bacterium. On another part of the whiteboard, draw the number 1.

1

Let us say that it has more than enough food. Then it will divide in two, and there will be two. Draw a second circle nearby and change the number to 2.

2

Again, there is enough food, so each of those two divides into two. How many do we have now? Update the whiteboard to show.



4

Can they figure out how many there will be next? The answer is 4 x 2 = 8. You can see how one bacterium can become a very large number of bacteria very quickly.

Show them this video which demonstrates this effect (speeded up) with real microscopic footage of bacteria: http://www.cellsalive.com/ecoli.htm

Explain that next session they are going to build a simulation that shows how bacteria grow like this, but first they have to understand the Scratch concept of cloning, so they will do a much simpler project.

## Scratch Cloning Explanation

Explain that in Scratch, you can create a copy of any sprite, and it is called a clone. Clones are used for when you want many of the same type of sprite, and they all act the same way.

Scratch gives you several features for clones:

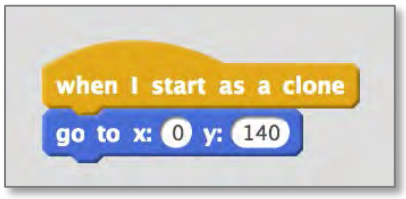
- Any sprite can create a clone of itself or another sprite
- You can have a script that runs when the clone is created
- You can delete a clone

## Lesson Procedure

Show the students a demonstration of cloning.

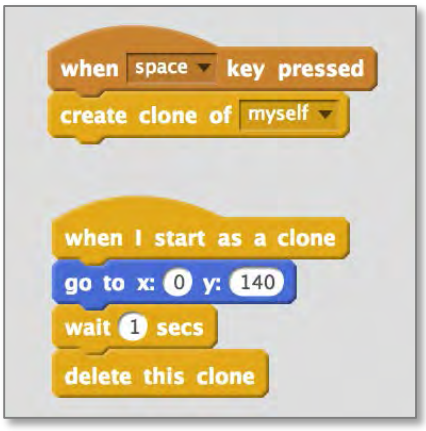
1. Create a new project. You can use the cat sprite this time for convenience.
2. From the **Events** category, drag "when space key pressed" into the code area.
3. From the **Control** category, drag "create a clone of myself" underneath it. Your code should look like this:


4. Press the space key. It appears that nothing has happened. Drag the cat from its current position, and you will see that you have two of them now. The reason you did not see two at first is that when the clone is created, it is exactly the same, including the position. So there were two cats, but they were right on top of each other. Let us add code to move it when it is created.
5. Click the stop button. This will delete any clones.
6. From the **Control** category, drag "when I start as a clone" into the code area.
7. From the **Motion** category, drag "go to x:_ y:_" underneath. Change x to 0 and y to 140 so that it will go to the top center of the screen. Your added code will look like this:




8. Now press the space bar, and two cats will appear.

9. Click the stop button again. From **Control**, add “wait 1 sec” and then “delete this clone”. Your total code should look like this:



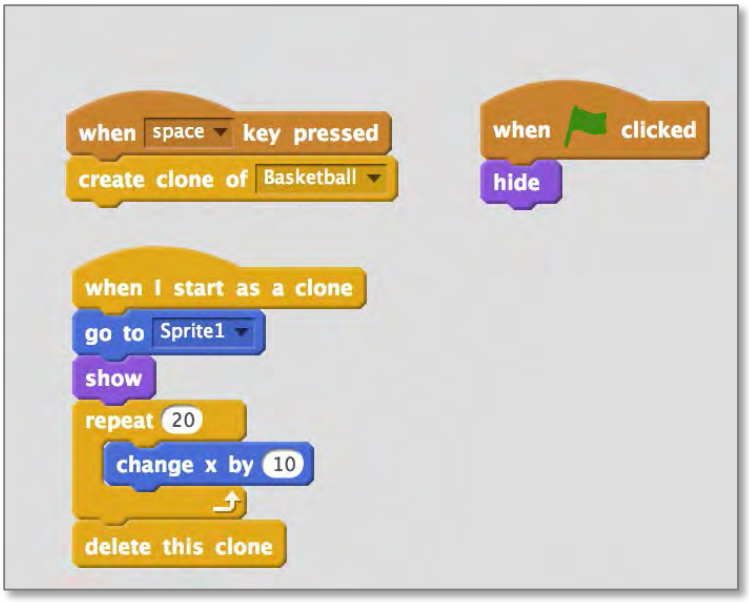

10. Now when you press the space bar, the second cat will appear for 1 second and then disappear.

## Cloning Project

Have students create a new project where they will make the cat throw a basketball. When the space bar is pressed, a basketball sprite should be cloned, go to where the cat is, then move 10 in the x directions for 20 times, and then delete itself. See if they can figure it out. It involves some blocks that you have not talked about yet (“go to sprite” and “repeat”), but are not difficult to figure out. Ask advanced students to help struggling students.

Here is what the final scripts should look like on the basketball sprite:

Advanced students can use the "hide" and "show" blocks in the **Looks** category so that the basketball is hidden to start with, and then is shown when starting as a clone. This way the ball only appears when the cat is throwing it. See if they can figure out how to do it. The code can look like this:




## Practice Activity Check

At the end of class, ask students to explain:

- What did you learn about how bacteria multiply?
- How do you clone a sprite in Scratch?

Please check the students' explanation for accuracy.

## Rubric

For the cloning project, use the Rubistar Website Design rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=29&section_id=3

Use the rubrics: Graphics, Layout, Content Accuracy (is it a good simulation?), and Learning of the Material.

# Lesson 9: Bacteria Simulation 2

Middle school: Skip this lesson
High school: Session 7

## Summary

Using information from the previous session, students are led through building a simulation that demonstrates how bacteria multiply. They will graph the number of bacteria for each generation, and show how the number plateaus when the food supply is limited.

## Learning Objectives

After doing this session, students should be able to:

- Create a simple model of bacterial growth
- Create a visualization for that model
- Understand how Scratch sprites have an inherent direction
- Stop scripts in Scratch

## Materials

- A computer for each student with internet
- Teacher's computer with internet
- Projector
- Whiteboard or equivalent
- 2 sheets of graph paper per student
- Pencils

## Preparation

Read through all of the sections so that you make sure you understand them and go through the steps in the **Lesson Procedure** sections and make sure you can do them. You can see a version of the bacteria project here:

Bacteria: https://scratch.mit.edu/projects/68659998/

## Vocabulary

The following vocabulary is used in this lesson:

- **Direction** – In Scratch, a value from 0 degrees to 359 degrees that indicates in which direction a sprite is pointing, and also the direction it will move with a "move" block.

## Introduction

In this session, you will lead the students through creating a simple model of how bacteria grow. This model uses two features of Scratch that you have not discussed before:

1. The idea that all sprites have not just a position, but also a direction.
2. Scripts can be stopped from running under certain conditions

You will also show how a simulation can be used to gather data. In this simulation, students will graph the number of bacteria as a function of generation number. The first version of the model will demonstrate exponential growth, since the bacteria double each generation. The second version provides a limited space with food, so the number of bacteria can only grow to the level that food can support them.

## Review

Quickly review what they learned in the last session. Show the video again to remind them how bacteria grow: http://www.cellsalive.com/ecoli.htm

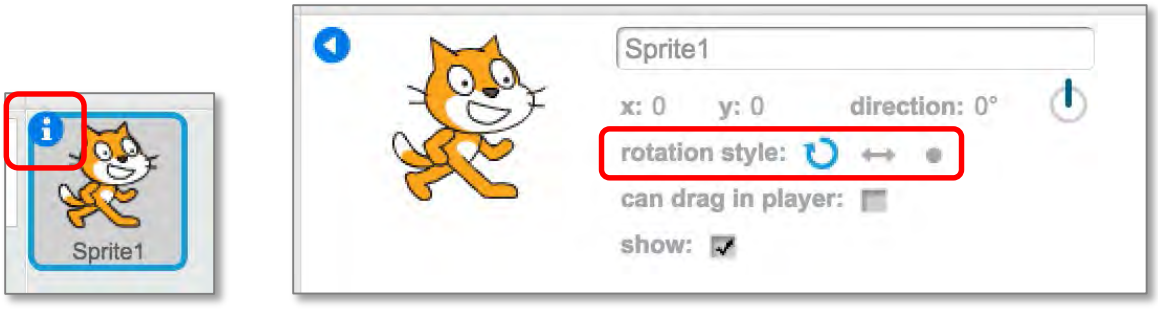
## Scratch Sprite Direction

This is a simple concept, so it does not require much explanation. Simply point out parts of the Scratch user interface.

Each sprite in Scratch has a direction it points in. The direction is described as an angle in degrees. 0 degrees is up, 90 degrees is right, 180 degrees is down, and 270 degrees in left. You can also have numbers in between.

Show the students the **Motion** category. To change the direction, they can use either the "point in direction" blocks or the "turn __ degrees" blocks. Then, the "move" block will move the sprite in whatever direction it is pointing.


Note also that the sprite image will rotate when you change the direction. If you do not want it to do that, then click on the icon near the sprite and choose a different rotation style. The first rotation style has the sprite image rotating with the direction. The second one has the sprite image only looking left and right. The third one has the sprite image always the same, regardless of direction.




## Model Discussion

Discuss how you will create a model for this bacterial growth. On the whiteboard, write down these rules:

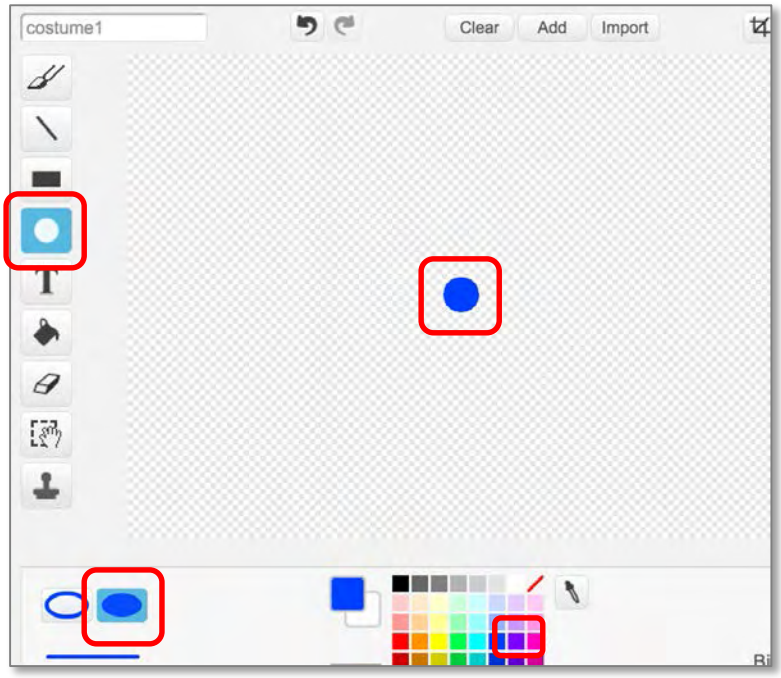
1. Every generation, each bacterium splits into two bacteria.
2. Once created, the bacteria move until they are no longer touching another bacterium.

## Lesson Procedure 1

Demonstrate the following Scratch project to the students. Afterwards, they will build it themselves. Follow these steps:

1. Create a new project in Scratch.
2. Delete the cat.
3. Create a new sprite, but this time draw it rather than use the library. Click on the paintbrush in the new Scratch area.

Choose the circle tool and the filled circle, and then the color blue. Look for a faint cross in the middle of the screen. Click and drag to draw a small circle that is centered approximately on that cross. You can hold the shift key down as you drag to make a perfect circle.

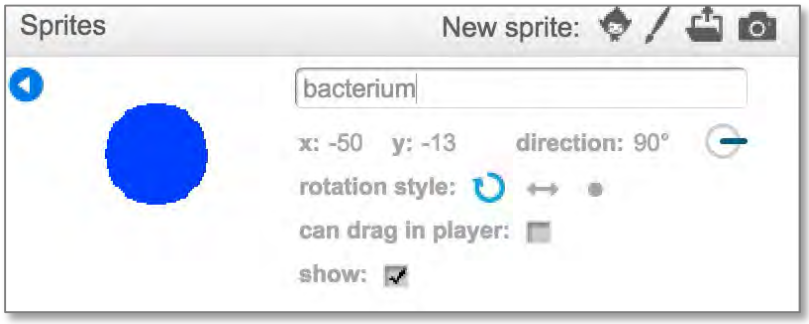



4. The circle should be a little bigger than the arrow cursor. If it is too big or small, use the grow or shrink buttons.

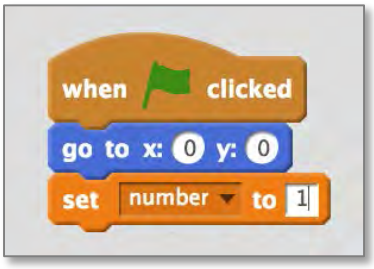

! **Important:** If the circle is too small, then it will result in too many bacteria being created, which will overwhelm Scratch. If there are too many clones created, then Scratch will simply not create new clones, which will mess up the numbers calculated. Even though small circles are more realistic, given how small bacteria are, you will get better results if the bacteria size is a little bigger than the arrow cursor.

5. Click the **Scripts** tab.
6. Let us create a variable to store the number of bacteria. In the **Data** category, click **Make a Variable**.
7. Name the variable “number”. Leave it as “for all sprites”.

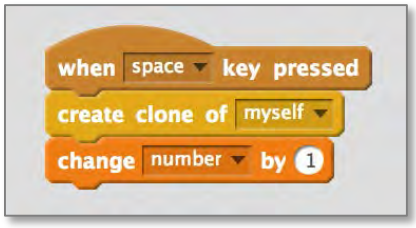
8. In the Sprites section, click the button. Then change the name to “bacterium” (or “bact”, if you want something shorter). It is a good idea to name your sprites. It makes your code easier to read.



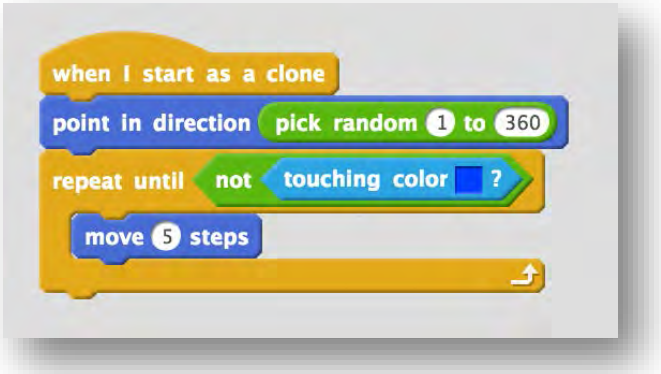

9. Let us start off with one bacterium in the center. From the **Events** category, drag out “when green flag clicked”.
10. From the **Motion** category, drag out “move to x:__ y:__”. Make them both be zero. This will put the sprite in the center of the screen.
11. From the **Data** category, drag out “set number to 0”. Change the 0 to 1. We are starting with one bacterium. Your code should look like this:




12. Next, when the space bar is pressed, let us advance one generation. From the **Events** category, drag out “when space key pressed” into a new area.
13. Point back to the model on the whiteboard: each bacterium will duplicate itself. From the **Control** category, drag out “create a clone of myself”.
14. Next, we need to increase the number of bacteria. Under that, from the **Data** category, drag out “change number by 1”. Your new code should look like this:



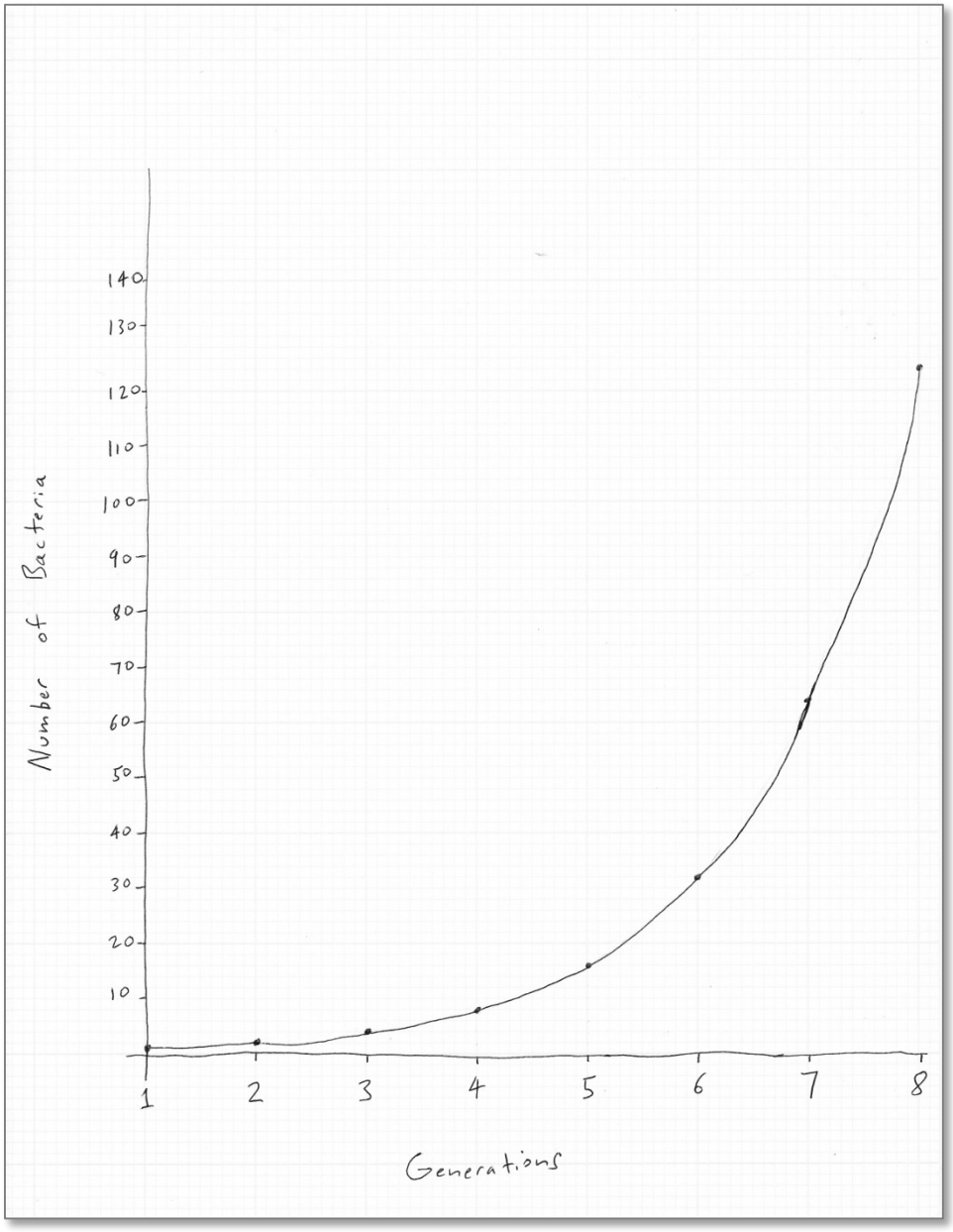
15. Now you just need the second rule of the model. Let us have the new clone point in a random direction and then move until it is not touching another bacterium. From the **Control** category, drag out “when I start as a clone” and put it in a new spot on the code area.
16. You will want to turn the sprite to point in a random direction. From **Motion,** drag out “point in direction 90” and put it under “when I start as a clone”.
17. From **Operators**, drag out “pick a random number” and put it where it says “90”. Change it from “1 to 10” to “1 to 360”.
18. Now, let us move in a random direction until we are not touching blue. From **Control**, drag out a “repeat until” loop. This will repeat until the statement in the block is not true.
19. We want to stop if it is *not* touching any bacteria, so we need a “not” statement. From **Operators**, drag out a “not” and put it into the “repeat until” block.
20. From **Sensing**, drag out “if touching color” and put it inside the “not” block. Click on the square and then click on the bacterium to make the color blue.
21. From **Motion**, drag out a “move 10 steps” block and put it in the “repeat until” loop. Change 10 to 5. Your new script should look like this:




22. Click the green flag. You will see one circle. Press the space bar, and you will see two. Press it again, and you will see four. You will see the program slow down considerably when you get to a large number of bacteria.

## Data Measuring 1

Pass out 2 sheets of graph paper to each student and make sure students have pencils. Give them several minutes to reproduce the project that you just demonstrated. Then have them graph the number of bacteria for each generation. They should label the axes, which you can show them on the whiteboard. Have them stop at 128.(After that it gets really slow.) The graph should look like below:



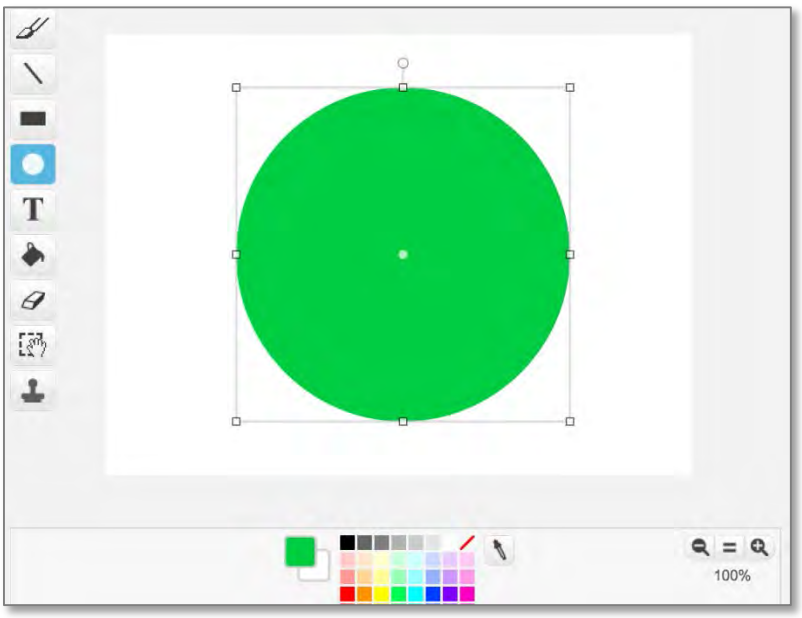
## Lesson Procedure 2

Return to the Scratch project and now add an area that will serve as food. With limited food, how will the bacteria grow? Add the following rule to the model:

| |
|---|
| 3. If a bacterium leaves the food area, then it dies. |

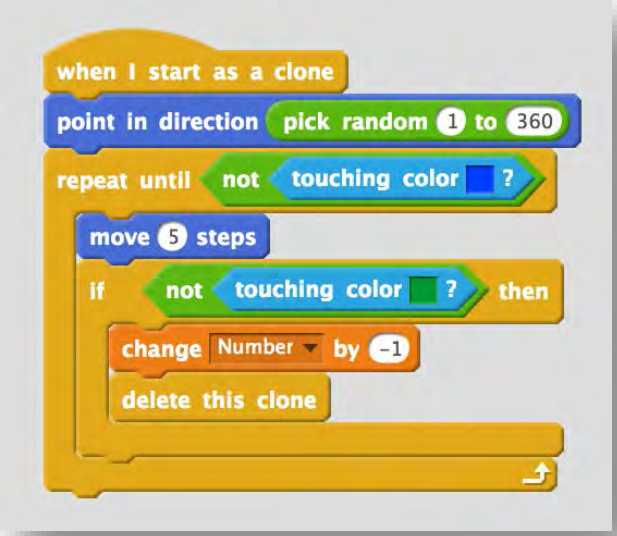
Follow these steps to add this rule:

1. Open up the Scratch project with the bacteria.
2. Click on **Stage** and then the **Backdrops** tab.
3. On the backdrop, draw a large green circle to represent the food area. (You draw the circle the same way you drew the bacterium circle.)




4. Click on the **bacterium** sprite in the Sprites area and then on the **Scripts** tab. You should see your scripts from before.
5. You need to add the blocks for the rule, "If you are not touching green, then delete yourself". From the **Control** section, drag in an "if then" block.
6. Right-click on the "not" of "not touching color" and choose **Duplicate**. This will duplicate the "not touching color ■". Drag it into the "if then" block. Click on the new blue square and then click on the green food area to turn it green.
7. For this new block of code, drag the "stop this script" block into the blocks area to delete it. (Or, right-click it and choose **delete**.)
8. First thing to do is to subtract 1 from the number of bacteria, since there will be one less once this clone is deleted. From **Data**, drag "change number by 1" into the empty "if/then" block. Then change the 1 to -1.

9. From **Control**, drag "delete this clone" into the "if/then" block. We have just added the code, "if you are not touching green, then delete yourself." This is rule #3. Your code should look like this:




10. Click the green flag. At first, it looks just the same as before, as long as all the bacteria stay on the green circle. However, after a few generations, they start to leave the green area and disappear.

## Data Measuring 2

Have them make a similar graph as before. They need to wait for all of the bacteria to stop moving in order to see the final number, and this can take a few seconds. This time, they will notice that the bacteria number grows to a certain level, and then does not get any higher. What this final value is depends on the size of the bacteria and the food, but it can be as high as 150, so be sure that the graph can go that high. This shows that a limited amount of food can support a limited amount of bacteria.

If there is still time left over, have the students try changing the shape of the bacteria to see how it affects the final number that the food can support. Can they make it look like the shape of the bacteria in the video? They can also look online to find other common bacteria shapes. See if they can figure out how to create new costumes for the sprite so that they can easily switch back and forth.

## Practice Activity Check

At the end of class, ask students:

- In Scratch, what does the sprite's direction mean?
- How do you stop a script in Scratch?
- What are the model and visualization for the bacteria?
- Did everyone finish the bacteria simulation?
- What changes did you observe to growth of bacteria by adding the food part of the simulation?

## Rubric

For the bacteria simulation, use the Rubistar Science Fair Experiment rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=1&section_id=4&

Use the rubrics: Data Collection, Diagrams, Display (quality of Scratch project), and Conclusion/Summary.

# Lesson 10: Energy and Economics Simulation

Middle school: Skip this lesson
High school: Session 8

## Summary

Students discuss simulation games that involve money, such as Farmville, SIMS games, etc. Then they play an existing Scratch game that is a simple energy simulation, involving choosing between adding coal plants and wind farms. You will show them the code and analyze how it works. Students then add a natural gas plant to the simulation, taking information from the web to use in their model.

## Learning Objectives

After doing this session, students should be able to:

- Research information on the web to be used in a simulation
- Create a simulation that has money in its model

## Materials

- A computer for each student with internet
- Teacher's computer with internet
- Projector
- Whiteboard or equivalent

## Preparation

Read through all of the sections so that you make sure you understand them and go through the steps in the **Energy and Money Simulation** sections and make sure you can do them. You can see versions of the projects here:

Original Energy and Economic Simulation: https://scratch.mit.edu/projects/68370640/

With added Natural Gas Plants: https://scratch.mit.edu/projects/68660084/

## Introduction

This session focuses on simulations where the user makes decisions by spending money and then seeing what they are able to achieve from their decision. This session is a little different from previous sessions because instead of creating a simulation from scratch, students will play with an existing energy and economic simulation, analyze it, and then add to it.

## Initial Discussion

Ask the students for examples of simulation games that use money. Examples include Farmville, SIMS games, Zoo Tycoon, Roller Coast Tycoon, etc. Have students explain how money is used in those games.

## Energy and Money Simulation

This is a very simple simulation that allows you to build coal plants and wind farms and then you get to see how much energy and carbon dioxide ($CO_2$) they produce.

**Note:** This very simple simulation does not even take into account the cost of the coal that is being burned.

### Play the Simulation

Follow these steps to demonstrate the simulation. Then have the students try it.

1. In a browser, go to this address: https://scratch.mit.edu/projects/68370640/
2. Note that energy is in megawatts, money is in million US dollars, and $CO_2$ is in billion pounds.
3. Click the green flag to start.
4. Note that every second, 0.1 years passes. You have 5 years to get as much energy as you can.
5. Hold your mouse over part of the screen and type "W". This buys a wind farm and puts it at that location. Note that your money goes down a little and you start producing energy. You are not producing $C0_2$.
6. Hold your mouse over part of the screen and type "C". This buys a coal plant and puts it at that location. Note that your money goes down a lot and you start producing a lot of energy. Also, you start producing $C0_2$.
7. As you produce energy, you also get more money to spend on wind farms and power plants.

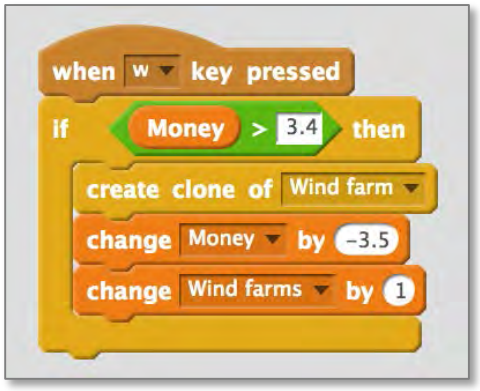
Have the students go to that web address and play the simulation. It takes about a minute. Ask them to read off what they ended up in for money, energy, and carbon dioxide. Have them explain their strategy. Then have them try it one more time.

### Analyze the Simulation

Click on **See Inside**. Note that the script for the coal plant is very simple: when I start as a clone, go to the mouse pointer. The script for the wind farm is exactly the same. So where is the model with all the equations?

Click on the **Stage**. This is where the code for the model is. Let us go through it piece by piece.

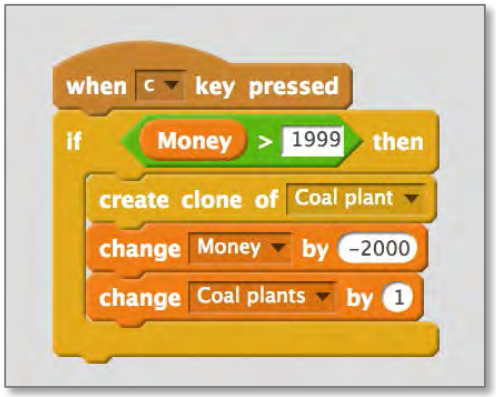
When the “w” is pressed, a wind farm is created. Here is the code:




With the students, analyze each block:

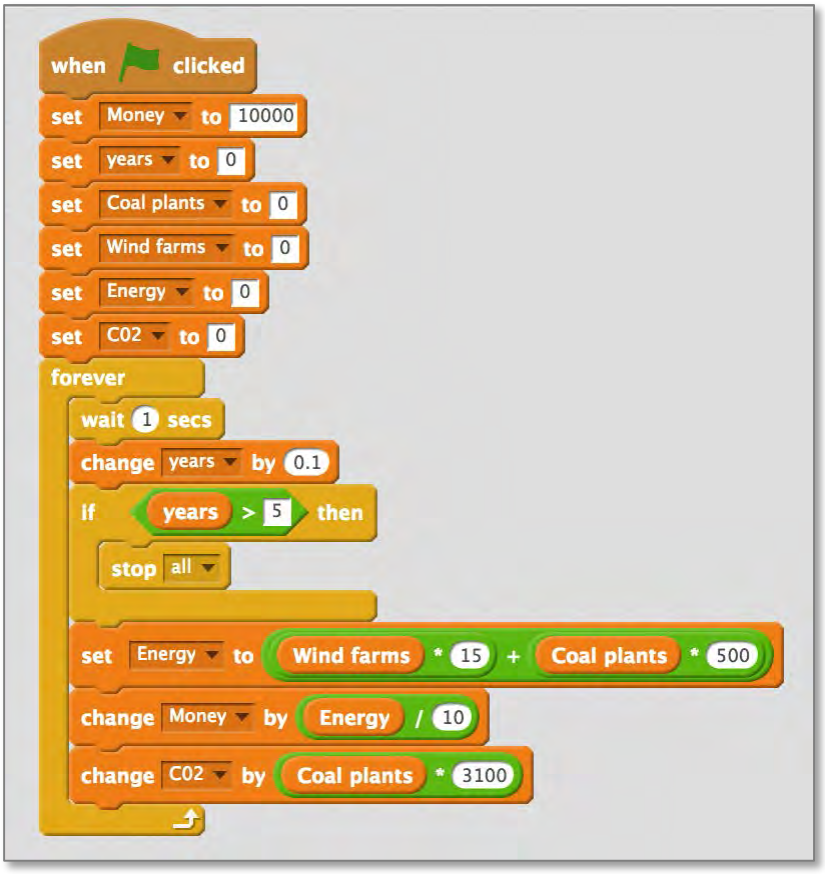
1. “when w key pressed” means that this script will be run when the w key is pressed.
2. “if Money > 3.4 then” means that it checks to see if there is enough money to buy a wind farm (3.5 million dollars). If there is, then it runs the blocks inside the “if/then” block.
3. “create clone of Wind farm” creates a clone of the Wind farm sprite. Remember, once it’s created it automatically moves to wherever the cursor is, because that code is in the sprite.
4. “change Money by -3.5” means to remove 3.5 from the Money variable, because a wind farm costs 3.5 million dollars.
5. “change Wind farms by 1” means to add 1 to the Wind farms variable, since we have just added a wind farm.

When “c” is pressed, a coal plant is created. Here’s the code:




It is exactly the same as the wind farm code, except it costs 2 billion dollars (2000 million) to build a coal plant.

The other script contains more equations:




Again, analyze the blocks with the students:

1. This script starts with “when green flag clicked”, so it is what happens at the start of the simulation.
2. The first group of “set” blocks starts you out with 10,000 million dollars, which is 10 billion dollars. The years, number of coal plants, number of wind farms, energy, and carbon dioxide are all set to zero to start with.
3. A forever loop is used to move forward in time.
4. Each time the loop is run, it waits 1 seconds and then changes the year by 1/10 of a year.
5. Once the year gets above 5 years, then the simulation is stopped.
6. The energy that is generated for each tenth of a year is 15 megawatts for a wind farm and 500 megawatts for a coal plant. So the coal plant generates a lot more.
7. Each tenth of a year, you get $1 million / 10 ($100,000) for every megawatt of energy generated.
8. The coal plants also generate 310 million pounds of carbon dioxide every tenth of a year.

There are also some yellow blocks. These are comments. The computer ignores them, but they are useful to help to someone looking at the code. They are being used to explain the units on the variables.

## Add to the Simulation

Ask the students to add a natural gas plant to the simulation. The first thing they need to do is figure out how much energy a natural gas plant produces and what it would cost. Divide into groups of 3 or 4 students and have them search the web on "natural gas power plant cost" and see if they can find this information. (If they get stuck, see if they can find an article called "Duke Energy natural gas-fired power plant may cost 18% less than planned". It has an example where a 625 MW plant costs $550 million.)

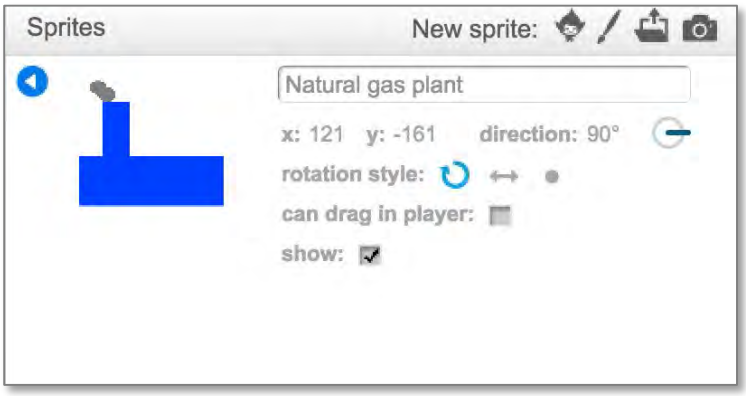
You can have them search for how much carbon dioxide a natural gas power plant produces, but it is tricky because most of the information on the web is in different units than what you need. Rather than spend a lot of time on this, you can give them this data:

Natural gas power plants create approximately 3.4 million pounds of $CO_2$ over a year for each megawatt generated. (See Appendix A for information on how to calculate this number.)

So, if you are using a 625 MW plant, you will generate 3.4 x 625 = 2,125 million pounds of $CO_2$ per year, which we can round to 2,100 million pounds per year.

Show them these steps to create the natural gas sprite.

1. In the sprites area, right-click the coal plant and choose **duplicate**.
2. Click on the **i** by the Coal plant 2, and change the name to "natural gas plant". Then close the sprite box by clicking on the blue circle with the triangle.
3. Click on the **Costume** tab and modify the image. Maybe color the plant blue or change the smoke or the shape.

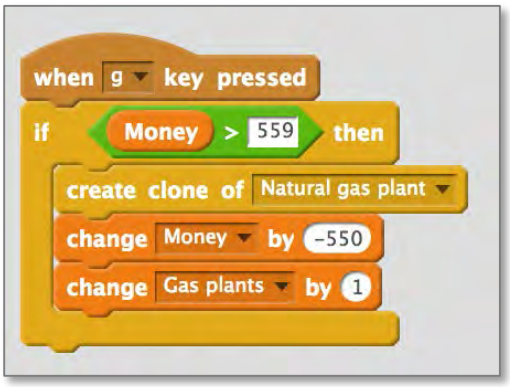


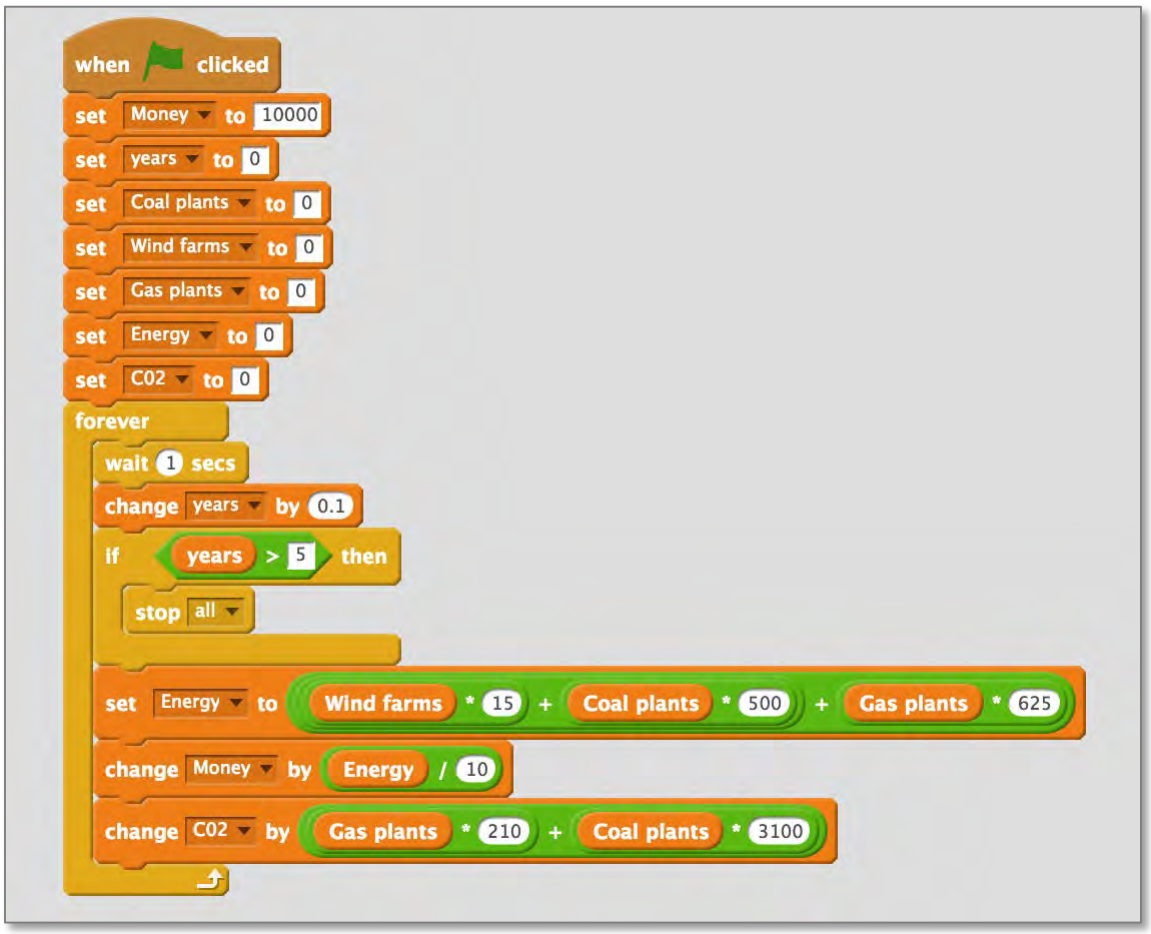

Now change the stage's scripts to add the natural gas plants to the model. They will need to:

1. Create a new variable called "Gas plants" that has the number of gas plants.
2. Duplicate the "when w key pressed" code and modify it so that when a certain key is pressed, a natural gas plant sprite will appear. Change the code so that it is correct for the price of the gas plant and that it increases the number of gas plants.
3. Add the natural gas plant data into the energy and CO2 formulas. The added energy will be the number of natural gas plants times the wattage of the plant. The CO2 will add the number you calculated above, but remember to divide it by 10 because each loop in the forever block is only 1/10 of a year. So for the 625 MW plant, you should multiply the number of gas plants by:

2,100 / 10 = 210.

Here's what the two scripts should look like when done:




Have the students try it out.

If there is still time, have students add a geothermal power plant. It will not have CO2 emissions. They will need to find the power plant costs and energy numbers on the web.

For advanced students: have them add the cost of the fuel for the coal and gas plants.

## Practice Activity Check

At the end of class, ask students to explain:

- How can you get information from the web to use in a simulation?
- How can you use money in a simulation?
- Check that everyone finished adding the natural gas plants to the simulation?

## Rubric

For the simulation, use the Rubistar Website Design rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=29&section_id=3

Use the rubrics: Graphics, Layout, Content Accuracy (is it a good simulation?), and Learning of the Material.

# Lesson 11: Choosing a Simulation Project

Middle school: Session 8
High school: Session 9

**Note:** High school students will skip the Economic Simulation part of this lesson

## Summary

Students will work in groups to brainstorm and choose an idea for their simulation project.

## Learning Objectives

After doing this session, students should have:

- Worked in groups to brainstorm simulation ideas
- Chosen a simulation to work on

## Materials

- Blank paper
- Pencils

## Preparation

Read through the lesson and choose a brainstorming technique.

## Introduction

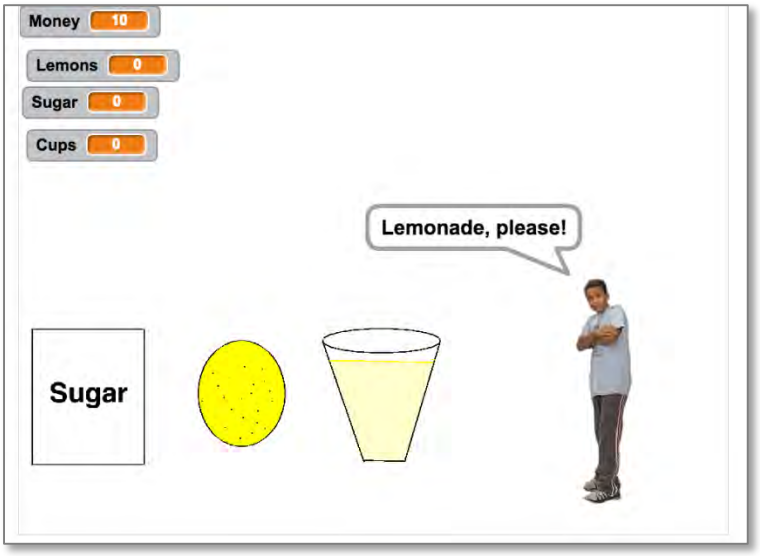
From this point onwards, the sessions are working sessions where students work in groups to choose, design, and build their final project. This session involves forming groups, brainstorming, and selection.

## Economic Simulation

One type of simulation that has not been covered yet for middle school is simulation that involves money. Ask the students for examples of simulation games that use money. Examples include Farmville, SIMS games, Zoo Tycoon, Roller Coast Tycoon, etc. Have students explain how money is used in those games.

Show them an example in Scratch. In this simple lemonade stand game, you must buy sugar and lemons, and then make lemonade fast enough to serve your customer.

https://scratch.mit.edu/projects/68370232/




Demonstrate the game:

1. Click the green flag.
2. Quickly: click the sugar box once to buy 10 tablespoons of sugar. Then click the lemon to buy a lemon. Then click the cup to make lemonade.
3. Every 2 seconds the customer will ask for a cup of lemonade. If the cups variable gets down to zero, and you cannot sell him any, then you lose.
4. Every time you make a cup of lemonade, it takes half a lemon and 1 tablespoon of sugar, so you have to watch those numbers and buy more when they are low.
5. Every time the customer buys a cup of lemonade, you make $1.

So this is another type of simulation that they can do.

## Forming Groups

Students should form groups of 2 to 4 people to work on the final project. The ideal number is 3 students. The process of forming groups is up to you, and will vary depending on the class. Some teachers have found that groups where students have similar life experiences (that is, come from the same culture, same sex, same age, etc.) often work better than mixing students from different backgrounds. Although integration of students with different backgrounds is a good goal, under many circumstances, this can result in certain voices not being heard.

## Idea Generation

The groups then use a brainstorming process to come up with ideas for their simulation project. You can use any brainstorming process you like. This section describes one particular brainstorming process.

Students may naturally want to jump straight into a solution. You may want to emphasize that all great apps solve a problem. It is important to understand what that specific problem is and only then consider the solution.

As with most brainstorming processes, the point is not to analyze or eliminate ideas or problems right now but rather to list them all without judgment.

Idea generation is one of the hardest and most important parts of the process of making anything. For this reason, you may want to explore other processes if the process below is not right for you or your students in any way. PBL Tech offers more options for brainstorming, including mind mapping and other methods: http://pbltech.wordpress.com/category/project-process/brainstorm/.

### Procedure

1. In small groups, participants sit around a table and each writes one area that interests them on the top of the sheet. It needs to be something physical, rather than electronic. Examples: space, farming, fashion, animals, etc. Electronic objects like video games are already a simulation, so they are not a good choice.
2. At your signal, each participant has 2 minutes to briefly describe one or more things that could be simulated in that area of interest. Just like in traditional brainstorming, the ideas should always go unedited. The difference is that now they are being recorded in private. The number of ideas and duration can vary, but aiming for one problem every thirty seconds works well.

   Here are some examples of ideas:
   - Space example: Planetary orbits
   - Farming example: Crop rotation
   - Fashion example: Factory for clothes
   - Animals: Select traits to breed animals for

   When time is up (or when everybody is done), each participant passes the sheet of paper to the team member to their left.
3. Each participant now reads the ideas that were previously written for 30 seconds and a new 2 minute round starts. Each participant must either come up with a new idea OR continue by extending an existing description that is already on the sheet.
4. Repeat. The group can agree to stop after a fixed number of rounds (such as when sheets come to a full turn around the table) or when participants feel that contributions are exhausted.
5. After the idea-gathering phase is completed, the ideas are read, discussed and consolidated with the help of the moderator, just like in traditional brainstorming.

## Idea Selection

Groups then choose one idea and refine it. They can choose the idea through informal discussion or they can use a technique called “dot voting”:

1. Place all the ideas recorded during idea generation up on a wall or white-board.
2. Each member of the team individually places a dot next to all the ideas they feel really passionate about taking forward. They have a total of 3 dots that they can place.

What makes a good simulation? A good simulation should (1) be based on something in the real world and (2) allow you to make changes to the model and see what happens. Students may come up with Scratch projects that are not based on something in the real world, in which case they are simply video games. Alternatively, they may come up with something that does not allow changes to the model, in which case they are simply presentations of information (for example, a slide show of scientific illustrations would not be considered a good simulation). Encourage them to choose a simulation that has both of these features.

## Review

Some of the ideas that the students have come up with may be quite complex. Review each group's idea and help them choose simulations where the model is manageable in the time that they have. Often, you can help them come up with a variation that is simpler. They will want to create something about the complexity of the simulations they have built so far, or perhaps a little more complex for the advanced students.

## Practice Activity Check

At the end of class, ask each team:

- How did the brainstorming go?
- Did every group choose a simulation project?
- Is the simulation reasonable to create in the time that you have?

## Rubric

For the simulation idea, use the Rubistar Video-Preproduction rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=109&section_id=3

Use the rubrics: Teamwork and Concept

# Lesson 12: Creating the Model

Middle school: Session 9
High school: Session 10

## Summary

Students will work in groups to create a model for their simulation.

## Learning Objectives

After doing this session, students should have:

- A set of rules and equations that serves as the simulation's model

## Materials

- Blank paper and pencils
- Alternatively, Google docs or Microsoft Word

## Preparation

Read through the lesson and understand what the students need to create.

## Introduction

The first step in creating a simulation is to determine the model, which will be similar to the models that you have shown the students in past sessions. Their model will not be "set in stone" – as they start working on their project. They may find that it needs to change.

If students end up creating the model quickly, then they can move on to the visualization.

## Creating the Model

The model consists of a list of rules and equations. Students need to figure out what data they will need to keep track of. (Money, number of animals, number of plants, velocity of a planet, etc.) and then write the rules and equations. Models should have at least 3 rules, but hopefully not much more than 10.

Students may need to look up information on the web for the equations. This is a good way to split up work among the students in the group. Be sure that they do not spend too much time on this.

### Example (for Middle School)

Here is an example of a model for lemonade stand simulation:

1. You start out with $10
2. It takes 1 tablespoon of sugar and 0.5 lemons to make a cup of lemonade
3. You get $1 for every cup of lemonade that someone buys
4. 10 tablespoons of sugar costs $1
5. 1 lemon costs $1
6. The customer asks for a cup of lemonade every 2 seconds

### Example (for High School)

Here is an example of a model for the energy simulation.

1. Users place coal plants and wind farms wherever they want to
2. Every second in the simulation is 1/10$^{th}$ of a year
3. Wind farms cost 3.5 million dollars
4. Coal plants cost 2 billion dollars
5. The energy generated in megawatts is 15 times the number of wind farms + 500 times the number of coal plants.
6. Every 1/10$^{th}$ of a year, you earn the energy/10 in millions of dollars.
7. Every 1/10$^{th}$ of a year, each coal plant generates 3100 million pounds of carbon dioxide.

## Practice Activity Check

At the end of class, ask each team:

- Did you successfully create a model?
- Do you have ideas for a visualization?
- Ask the students to spend time to rate and refine their ideas

## Rubric

For the model, use the Rubistar Making a Game rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=9&section_id=2&

Use the rubrics: Accuracy of Content, Rules, Cooperative Work, Creativity

# Lesson 13: Creating the Visualization Design

Middle school: Session 10
High school: Session 11

## Summary

Students will work in groups to create the design of the visualization for their simulation. This means that they will decide what sprites they need and draw a picture of what the screen will look like.

## Learning Objectives

After doing this session, students should have:

- A list of sprites that they will use
- A description of the background
- A sketch of the screen

## Materials

- Blank paper & pencils

## Preparation

Read through the lesson and understand what the students need to create.

## Introduction

Now that they have the model, students will work on the designing the visualization, which means creating a design for their Scratch project. This consists of two parts:

- Describing what sprites they will use, what they will look like, and a brief description of what code they will have
- Describing the background, both in terms of graphics and code
- An annotated picture of the screen

If students end up creating the visualization quickly, then they can move on to the building it in Scratch.

## Creating the Visualization

On a sheet of paper, students should list all of the sprites. Each sprite should have a sketch of what it looks like, and a one or two sentence description of what code it will have.

At the bottom of the sheet, they should add a sketch of the background and a short description of any code it will have (if any).

On a second sheet, they should sketch what the screen will look like and add labels that explain.

### Example (for Middle School)

Here is an example of a visualization design for the lemonade stand simulation.

## Sprites

Sugar 


When clicked, buys 10 tablespoons of sugar for $1

Lemon 

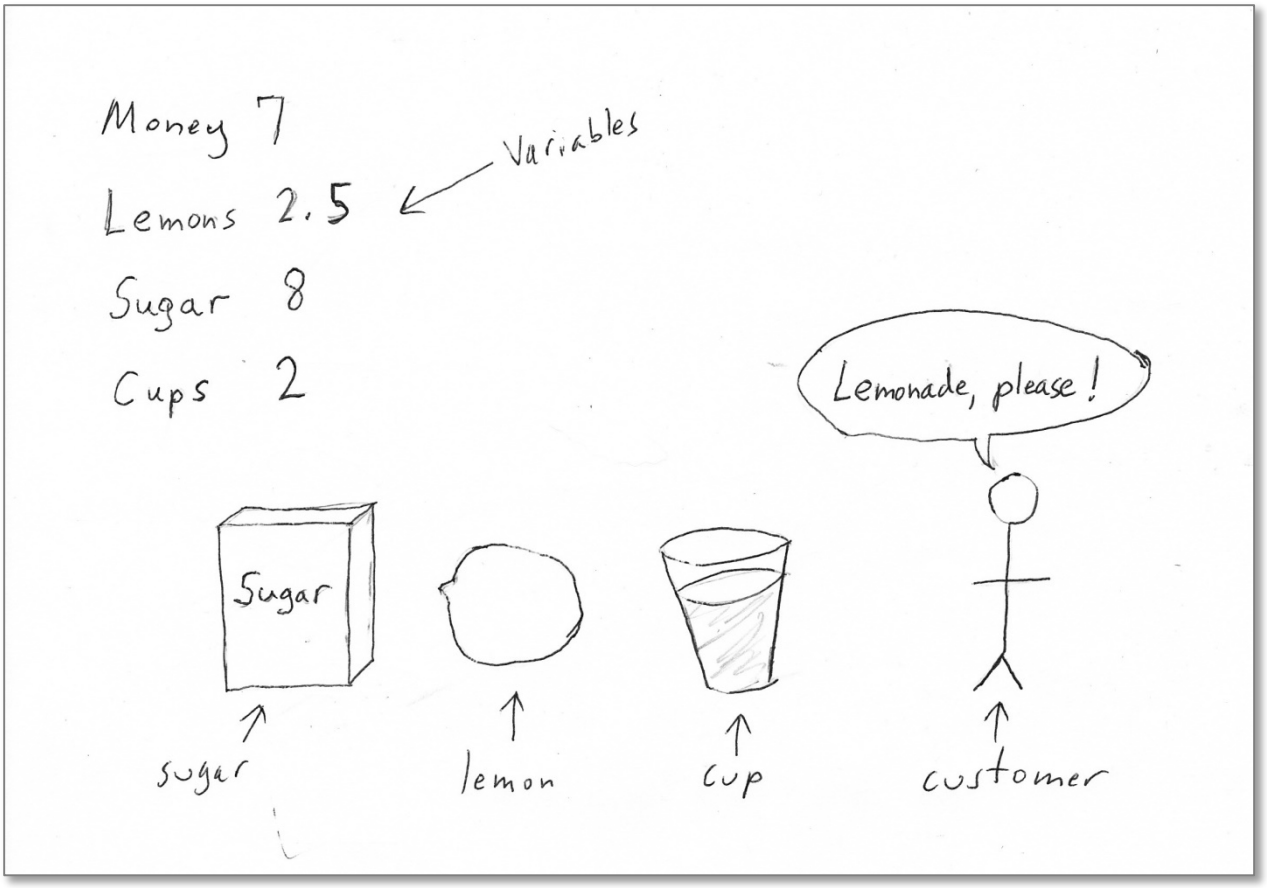
When clicked, buys a lemon for $1

Cup 

When clicked, makes a cup of lemonade from 1 tablespoon of sugar and 1/2 lemon

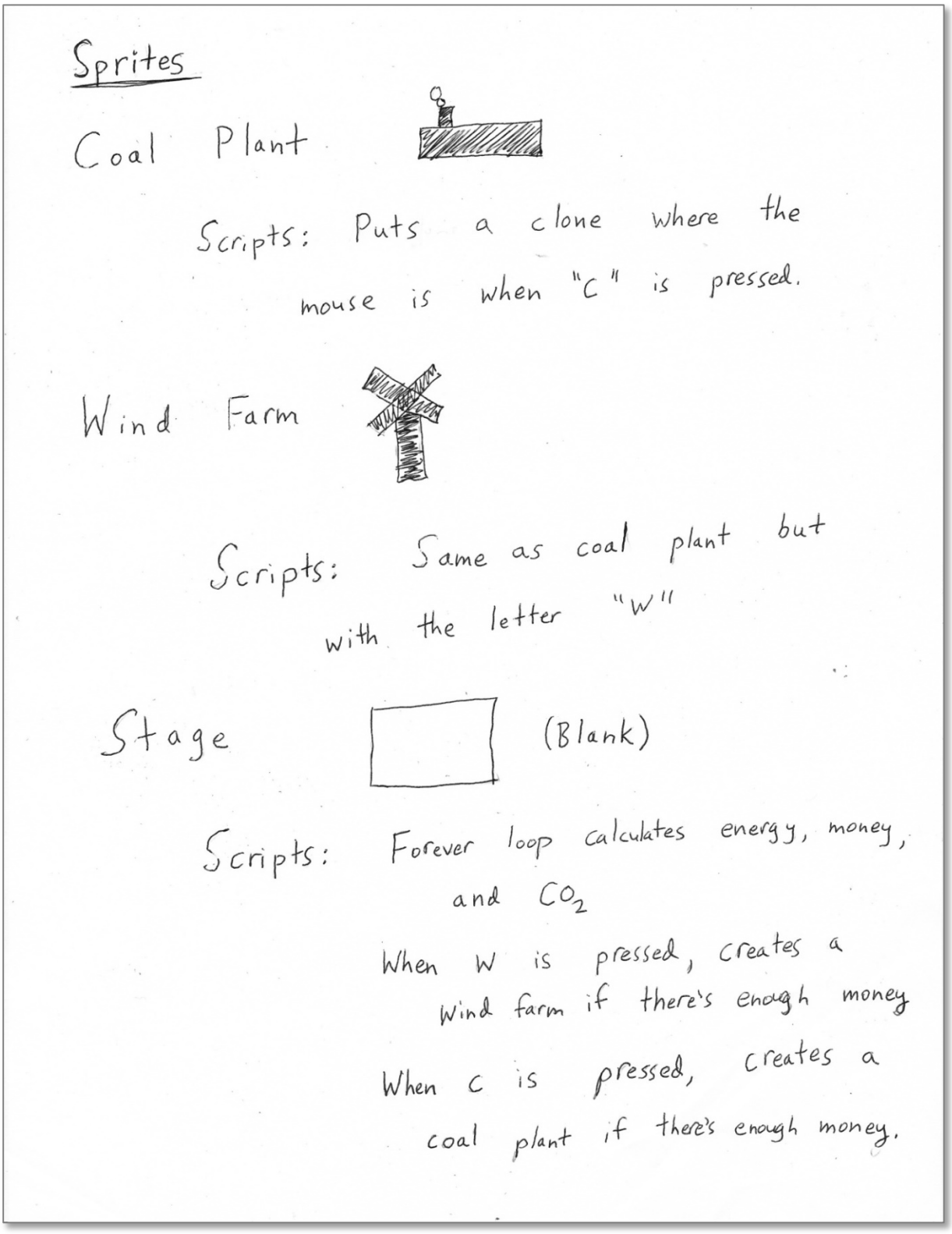
Customer 

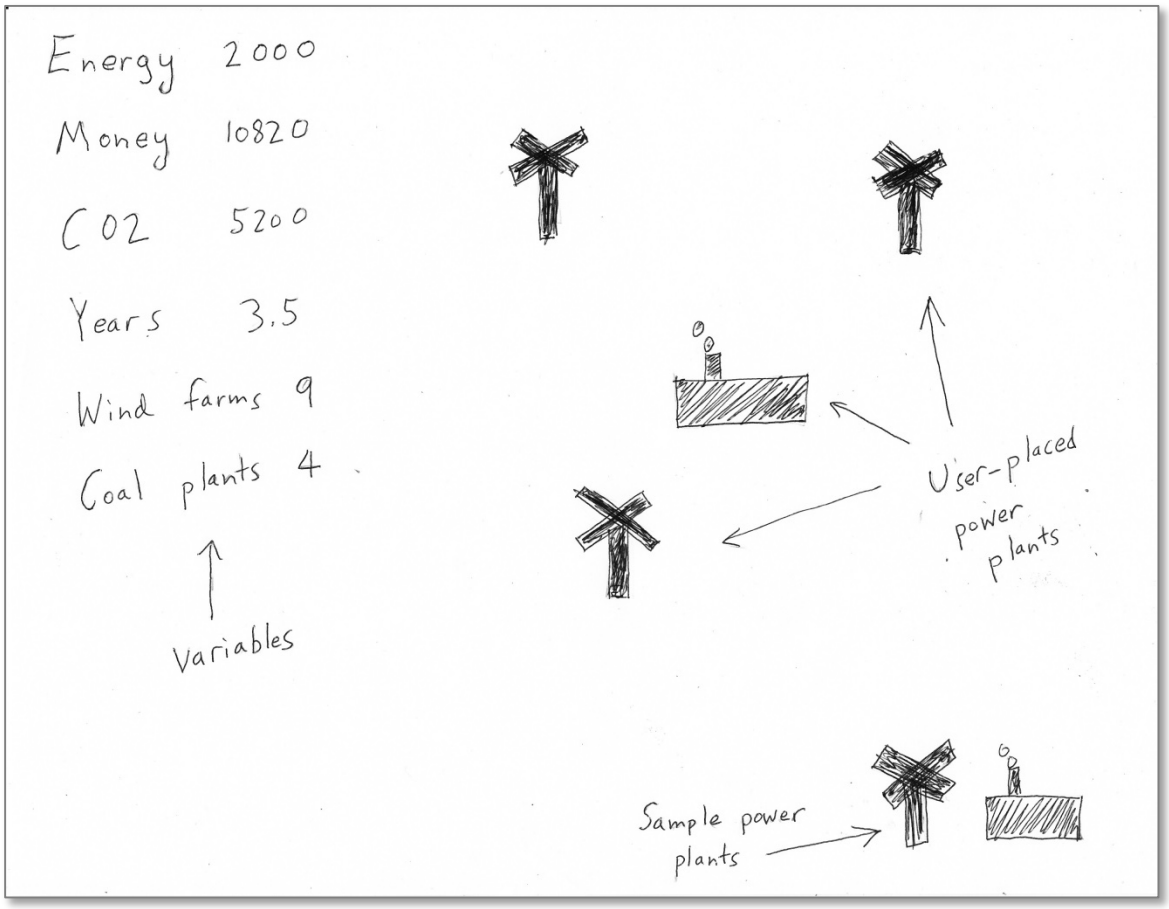
Every two seconds, buys a cup of lemonade for $1


Money 7
Lemons 2.5
Variables
Sugar 8
Cups 2
Lemonade, please!
Sugar
sugar
lemon
cup
customer

### Example (for High School)

Here is an example of a visualization design for the energy simulation.






## Practice Activity Check

At the end of class, ask each team:

- Did you successfully create a visualization design?
- Can you see how it will work with the model you created?

## Rubric

For the visualization, use the Rubistar Storyboard-multimedia rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=9&section_id=2&

Use the rubrics: Clarity and Neatness, Cooperation, Content, Required Elements

# Lesson 14: Building the Simulation

Middle school: Sessions 11-14
High school: Session 12-14

## Summary

Students will work in groups to build the simulation based on their model and design.

## Learning Objectives

After doing these sessions, students should have:

- A working version of their simulations

## Materials

- Computers with internet
- Scratch accounts

## Preparation

Read through the lesson and understand what the students need to create.

## Lesson

Students now work on their projects in their groups. Use the strategies described in the unit introduction so that you do not need to understand everything about Scratch.

In addition, here are some techniques that will allow multiple students to work together to build one project:

- Have one student start the project, which will act as the master project. Then have them duplicate it so that others can work on it and add features independently
    - If they are using the same account, use **Save a copy** from the **File** menu.
    - If they are using different accounts, have the first student click the **Share** button to make the master project public, and then have the other students go to the URL address of the project, and click **See Inside** and then **Remix**.
- To merge the features back together again, have the first student open the project with the new features and then drag sprites or code into the "backpack" at the bottom of the screen. Then have them open the master project and drag those elements from the backpack into the proper places in the master project.

Make sure that in addition to code, their project has instructions on the project page so that people can figure out how to use it.

## Rubric

For the simulation project, use the Rubistar Website design rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=29&section_id=3&

Use the rubrics: Graphics, Content, Layout, Content Accuracy (how well does it show the model?), Interest, Learning of Material, Cooperative Work

# Lesson 15: Testing and Improving

Middle school: Session 15
High school: Session 15

## Summary

Students will test each other's simulations and provide feedback. Teams will then make corrections.
**Note:** This session is optional. If students need more time for their projects, this session can be skipped.

## Learning Objectives

After doing this session, students should have:

- Provided thoughtful feedback on other students' projects
- An improved final project

## Materials

- Computers with internet connection
- Printouts of the Simulation Testing sheet (in this section)

## Preparation

Read through the lesson and understand what the students need to create. Make testing assignments so students know which simulations they will test.

## Introduction

Testing is an important part of any software project. It is important to have people who did not build the software to test it so that there's an outside point of view. In this session, each student will test 2-3 projects from other students.

## Testing

Have students test out each other's simulations and fill out the form on the next page. It can be difficult for students to articulate their testing results, so encourage them to be specific. They should test at least two simulations, but more if time permits.

## Fixes

Distribute the test results to the student teams. Have them discuss the results and see if there are any changes that they want to make. Often they will need to make improvements to the instructions.

## Practice Activity Check

At the end of class, ask students:

- Did you learn anything from testing?
- What did you improve?

## Rubric

For the testing, use the Rubistar Collaborative Work Skills rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=29&section_id=3&

Use the rubrics: Contributions, Quality of Work, and Attitude

## Simulation Testing

Project Name: _____

Tester Name: _____

What I liked about the simulation:

What I didn't understand about the simulation:

What didn't work:

# Lesson 16: Final Model and Presentation

Middle school: Session 16
High school: Session 16

## Summary

Students will create a final version of the model and then each group will present to the class what their simulation is and how it works.

## Learning Objectives

After this session, students should:

- Have a final version of the model
- Be able to publically explain and demonstrate their simulation

## Materials

- Computers with internet connection
- Projector

## Preparation

Read through the lesson and understand what the students need to do.

## Introduction

This is the final session, so students' simulations should be complete. For the MESA competition, they will need to turn in a document that describes their model as a list of rules and equations as well as the final Scratch project with instructions.

## Final Model

Have students review the model that they created in Session 11. Now that they have built their final project, are there any updates they would like to make? Have them type up a final version in Microsoft Word or Google Docs so that it can be shared on the projector.

## Presentation

Groups take turns presenting the simulation to the class. There are several pieces they need to cover, and can divide up the work among the students in the group. The presentation should cover:

1. Introduction: what the simulation is and why did they choose it.
2. Model: Display the model document on the projector and go through it.
3. Visualization Design: Bring up the Scratch project and explain what each of the sprites are
4. Demonstration: Run the simulation and show how it works

## Rubric

For the presentation, use the Rubistar Oral Presentation rubric-generator at:
http://rubistar.4teachers.org/index.php?screen=CustomizeTemplate&bank_rubric_id=4&section_id=1&

Use the rubrics: Comprehension, Enthusiasm, Content, Collaboration with Peers