LYNX™

ASTRONAUT VS ALIENS
TEACHERS' GUIDE

Astronaut vs Aliens - Teachers' Notes

Project Overview

Game-building is a rich learning experience that taps into a wide range of abilities and interests. To start with, it's creative! Students can let their imaginations go wild as they develop the narrative for their game action. In addition, it involves logic in how different playing choices are set up and the outcomes to which each choice leads, an understanding of visual impact and its effect on gameplay, and a strong awareness of audience and the user experience. Even the simplest games require considering all these factors.

In this project, students build an Astronaut vs Alien game, one of the best ways to launch into the world of gaming. They can be very simple yet fun, non-violent, have a range of different characters and circumstances, and appeal to all types of people. For an interesting explanation of the history of seeker games from the original seeker game of Pac-Man to all its offspring and their impact on today's video game landscape, check out https://www.fastcompany.com/1683023/how-pac-man-changed-games-and-culture.

In this project, the Seeker will be an Astronaut who needs to collect 3 solar panels to power-up their space ship and get back to Earth. Aliens have another idea and want to get the Astronaut.

This project consists of six 45-60 minute lessons that introduce a number of more advanced coding concepts such as conditional statements and logic operators, variables, and additional applications of event handlers (more on these below). At the same time, it reinforces skills practiced in previous projects.

Students will continue their work with student-created programs, interactive control objects (such as buttons and sliders), both collision (or "touching") detection and colour detection event handlers, several types of variables and the importance of designing an understandable graphical user interface (GUI). Finally, students will practice iterative incremental development, which may sound intimidating, but basically means ongoing program review and revision as students add new features and get new ideas about what they want to include in their games. It means having regular rounds of review, debugging, and revision.

Prerequisites

Before starting students should already be familiar with the following:

- The Lynx layout: **Work Area**, **Command Centre**, **Procedure Pane** and **Clipart Pane**.
- Basic turtle commands (for example, `forward`, `back`, `right`, `left`, `repeat`, `cg`, `setshape`, `stamp`, `setsize`, `xcor`, `ycor`) as well as any syntax associated with these commands.

- How to draw with the turtle (`penup`, `pendown`)
- How to work with **clipart**, **sound** and **text boxes**.
- How to define **procedures** (to, end, procedure name protocols, comments, and the use of sub-procedures).
- How to add and program **buttons** and **text boxes**.
- How to create **On touch** events.

Standards Addressed

In this project, students will meet the following learning standards:

- Plan and write more complex computer programs, applying fundamental programming concepts, and including clear internal documentation.
- Use conventions and terminology of computer science effectively.
- Use control structures in computer programs.
- Demonstrate the ability to use Boolean operators (e.g., `not`), comparison operators (for example, `equal?`, `member?`).
- Write programs that incorporate user input, processing, and screen output.
- Write sub-procedures that use parameter passing and appropriate variables (e.g., local, global) to perform tasks within programs.
- Demonstrate the ability to identify and correct syntax, logic, and run-time errors in computer programs and to interpret error messages displayed by programming tools.

Astronaut vs Aliens - Teachers' Notes

Before You Start

1. At this point you should already have your teacher account set up on Lynx and have rostered your girls in a team. In case one or some of your students are not registered with Lynx yet, give them some time at the beginning of Lesson 1 to create their own account by following the steps in the How to Create a Lynx Account:
2. In **All Projects**, inside the **Games!** Folder, look at the **Astronaut Game** sample at https://lynxcoding.club/ with your students to understand how it works and how it was created. Start by reviewing the following skills:
    1. How to describe turtle movements using commands.
    2. How to use clipart to add shapes for the turtle.
    3. How to use event handlers, specifically the **On touch** event handler.
    4. The use of words as a specific type of data in Lynx, as well as the appropriate punctuation.

Next, talk about the project features, encouraging students to think about the following:

1. What is the basic logic behind conditionals and why are “if” statements needed in games?

    All games make use of conditional statements, statements that first ask if something is true or false and determine what action happens based on the answer. Whether a card game, board game, or video game, conditional statements determine the flow of play and all decision-making outcomes. As students build their games, they will use a range of logic operators to modify their conditional statements. Logic in computer programming is not only important for gaming, but it’s at the core of many fields of computer science, including machine learning and other areas of Artificial Intelligence. Practice in using logic statements and operators prepare students for even more advanced programming.

    Suggest some examples from both everyday life and games they may have played. For example, in the card game war, *if my card > your card*, then I win. Or, *if your card = my card*, then we both must put down three cards upside down and turn over the fourth card. What other examples can students think of?

2. How can you use different logic operators, such as `not`, `and`, `or`, `member?`, and `equal?` to more precisely define logic statements? (Note: Although not all of these are used in this project, knowing these provide important options for students as they extend this project and create new ones.)

For example, how would the outcomes differ if someone says If you and your friend are older than 16, then you can both go to the concert, versus if you or your friend is older than 16, then you can both go to the concert.

Commands that end in a question mark are basically condition questions. Equal? asks if two things are equal (the same as the equal sign =, but sometimes easier to use depending on the elements of the conditional statement). Member? asks if one item (a letter or word, for example) is a member of another item (a word or group of words).

Can students think how they might use not?

Idea: if my age does not equal your age we are not twins (although even if they are equal, we may not be twins. But that's the interesting part of logic.)

3. Event handlers control responses to interactions such as when a sprite touches a colour or another sprite. What types of game features depend on or are enhanced by the use of interactive events?

   Have students first think of examples from real life. What happens when a person meets a stranger? Maybe nothing. But what happens when one friend meets another? What if one friend is angry at the other?

   Next, have students think of video games they play. What happens when one character jumps and hits (touches) another object in the game? What happens if one of the objects is a car and the other is a frog (as in the old game Frogger)? What if the character touches a "prize" object? What happens if it's an enemy?

4. How do you build an effective and attractive user interface (UI) or, in this case, graphical user interface (GUI – pronounced Gooey)? What makes a good interface?

   The GUI shapes how the user interacts with a game and is particularly focused on the game's graphical design – how it looks and how attractive it is. It takes into consideration the layout of all buttons, text boxes, sliders, and other features users may use to interact with the game. It also determines how and where user's get necessary information easily and in a timely fashion. The best game can be ruined by a confusing interface. Design helps lead the user through a game and enhances the user experience (UX). What's even more important is that a good GUI doesn't make itself noticed, but rather is an integral part of a well-made game. A well thought out, clean, clear, attractive, engaging design turns a good game into a great one.

   Most likely students have not focused on this aspect of the games they've played, so making them aware of how user interfaces are designed may make them more aware of

this important aspect of game building. Ask students to think about their favourite games and then share ideas on what they like in their GUIs and what they have seen that they don't like or found difficult to use.

5. Why use variables?

   Just as in mathematics, variables can stand in for values that may change over time. This makes it possible to give instructions to a whole group or class of objects. This makes it possible to write shorter, clearer code (grade 7 will leave at 3 PM versus Simon, Erin, Kale, Elijah, Layla, and so on will leave at 3 PM). It also provides a clearer picture of how one group of sprites may interact with another in the game.

- Have a conversation with your students about the importance of debugging and iterative incremental development. In this project, students implement the bare bones of their games and then gradually add and test new features. Highlight the importance of planning. Remind them to add comments to their procedures to help others understand their procedure use and logic.
- Remind students that Lynx ***does not*** automatically save their work so they must save their code often. Saving is quick and easy on Lynx by clicking on the **Save** icon (Arrow pointing-up to a Cloud) **AT Image**
- Students who choose to do the optional steps will be introduced to the use of ASCII (American Standard Code for Information Interchange) and how to use it to represent data and add interactivity. ASCII is a set of digital codes, in this case, numbers, representing letters, numerals, and other symbols, and is used as a standard format to facilitate the transfer of text between computers. For example, the ASCII code for the letter A is 65. You can use Lynx to find different ASCII codes, for example, type:

  ```
  show ascii 'a'
  97
  ```

  ASCII codes are different for capitals and lower case letters

  If you have an ASCII number and want to find out what character it represents, type:

  ```
  show char 65
  A
  ```

  Coding with ASCII codes is especially helpful when programming responses to keys such as the arrow keys or Return/Enter. Lynx can check if the ASCII code of a key the user presses is the same as a specific ASCII value of, for example, the up arrow. If it is some specific action happens, for example a sprite moves higher on the screen. This can be a very powerful tool in a student's game building toolbox.

Students may want to explore why this standard was developed, how it has changed over the years as computers moved from 8-bit characters or elements to 64-bit and how it compares to other standards, such as Unicode.

Learning Targets and Checklist

Projects should demonstrate the students understanding of the following skills. Each of these should contribute, in a logical way, to designing their game.

√	SKILL
	Write, edit, and debug more complex user-defined programs.
	Use commands to create multi-turtle animations.
	Add interactive control with buttons and sliders.
	Use both **On touch** and **On colour** event handlers to trigger specific actions under specific conditions for specific sprites.
	Use the word data type to create a personalized response to an event.
	Use if conditional statements and logic operators such as `not`, `or`, `equal?` and `member?` to determine actions at critical decision making points in their procedures.
	Create and use global, local, and state variables appropriately.
	Use list processing primitives (`first`, `butfirst`) to change variable values.
	Use the built-in timer to trigger actions.
	Practice iterative development and debugging skills as programming becomes more complex.
	Include clear comments in the Procedures Pane.

Useful Terminology

ASCII

ASCII (American Standard Code for Information Interchange) is a standard format designed to facilitate the transfer of text between digital devices. It is used to represent letters, numbers, symbols, and control keys such as space bar, arrow keys, etc, used to format text. Over time, the code has had some modifications and extensions, for example, to take into consideration special diacritical marks added to letters in various languages, but the main core and function of the code have remained the same.

Google ASCII and you can find a few web sites with a more complete explanation.

CONDITIONAL STATEMENTS

A conditional statement is an *if – then* logical statement. It first tests if a condition is true. If it is, then some specific action results. For example, if two numbers are equal (it's true), the word "match" is printed. An `ifelse` statement goes a bit farther. It not only specifies what to do if the condition is true, but also what to do if the condition is false.

GRAPHICAL USER INTERFACE (GUI)/USER INTERFACE (UI)

The GUI shapes how the user interacts with a game and is particularly focused on the game's graphical design. This includes the layout of all buttons, text boxes, sliders, and other features users may use to interact with the game. It also takes into consideration how and where user's get necessary information easily and in a timely fashion. A UI is more general and may include input devices (touch, mouse) and other ways in which a user might interact with a program.

ITERATIVE PROGRAMMING

Iterative programming is a programming process that involves multiple cycles of planning, development, and review, with new features added or new designs implemented in each cycle. It differs from a project in which all details are specified before development and review is only for debugging purposes, not for adding and/or shaping features and building up the actual design.

LOGIC OPERATORS

Logic commands are essential components of conditional statements and help define the condition being tested. They include commands such as `and`, `or`, `not`, and, in Lynx, any of the commands the end with a question mark, for example `key?` (is a key pressed?), `equal?` (are inputs equal?), and `member?` (is one item a member of another?).

New Concepts and New Lynx Commands

1. Preparing the background
   `setcolour`
2. Working with multiple turtles
   `clone`
   `ask`
   `everyone`
3. Randomness
   `random`
4. Parallel processes
   `forever`
   `stopall`
5. Collision detection
   `on touch event handler`
   `on colour event handler`
6. Using basic data types: words
   `word`
   `announce`
7. Reset procedure, buttons and UI elements
   `freeze`
8. Cartesian coordinates
   `setx, coorx`
   `sety, coory`
9. Conditionals
   `equal? if, ifelse, not, or, key?`
10. Timer
    `resett, timer`
11. Variables
    `make`
12. List processing
    `butfirst (bf)`
    `first`
    `member?`

Common Errors and Debugging

Debugging is an excellent medium for developing problem-solving skills. Lynx has a number of tools, to support students as they learn to code in Lynx, including auto-complete, tool tips, error messages, and, of course, a Help page.

To begin, it's good to identify the type of bugs a student may encounter. There are two main types of errors that programmers make. Both types of errors will occur at this level of programming. These will be explored and discussed in more detail below.

ERROR MESSAGES

It's also very important to remind students to pay attention to error messages, especially when they have many procedures and sub-procedures. They should particularly pay attention to the line number cited in the message if there's an error in one of the procedures.

TWO MAIN TYPES OF ERRORS

1. Typing or syntax errors

- Forgetting to put a space between a command and its input, like writing `fd30` instead of `fd 30`.
- Typos, for example typing `fe 20` instead of `fd 20`.
- Using the wrong type of input or syntax punctuation, for example, writing `repeat 4{fd 1 wait 1}` instead of `repeat 4 [fd 1 wait 1]`.

  When an error occurs, a message appears describing the error. Often, students ignore messages, but since these messages highlight where the error is, students should be reminded to pay attention to them. For example, if a student types `fd30`, they will get the message: `I don't know how to fd30`. Lynx interprets this single word as a new command that is neither a built-in command nor a procedure name. Unlike people, Lynx can't "interpret" or "figure out" what the student means, it can only respond to what is typed.

  This type of error can also occur in creating and using variables appropriately. If students are getting error messages, have them review the variables they created to make sure they've used quotation marks and colons appropriately

- Not adding `end` at the end of a procedure.

  Sometimes students forget to type `end` as the final line of a procedure. The procedure without the `end` line works normally, but the procedures written after that one won't be found. If a student is having a problem running a procedure from the Command Centre or from a button and gets the message `I don't know how to`, first check if all the procedures in the Procedures Pane include `end`.

2. Errors in logic

- Sequence of actions - Commands need to run in a specific order, depending on how the coder wants events to unfold. This idea of order is something everyone commonly deals with in life: we need to put on our socks before we put on our shoes and not the inverse. This algorithmic thinking is a main computational thinking skill.

  Although this seems easy, as programs become more complex, the sequence of events may become more difficult to follow. In this project, understanding the logic

should be fairly easy, but it's a good idea to help students be aware of how they determine what actions they may want to have happen before others and to be able to put their thinking into words.

- The flow of information from different types of commands, including those that send (or output) information to another command. These types of problems may appear as students begin coding more complex projects.

Final Notes

Each lesson should take about 45-60 minutes (although, if students have more time and want to continue, there's lots to explore!).

Urge students to spend time planning. This will help them get started and save them time as they go.

Remind students that although it's exciting to jump in and begin a new project, it's important to start by first focusing on their game's functional design. There are several things they should think about before they start coding. Remind students to:

1. **Know your goal!** The end project looks (and works!) better when you start with a plan and you know what your goal is.
2. Decide on some **project parameters**. How many prizes will the Astronaut have to find? What will the prizes be? How many lives will the Astronaut have? Who or what will be pursuing the astronaut? In the sample, the Astronaut is trying to find three solar panels but is also trying to elude aliens, but you should use your imagination.
3. **Create a first draft!** Start by creating a first draft of the project that includes the main project components. As a coder, it is important to plan and create your code before you spend time on the visual design features (Tip: Choose your final clipart last.) Go over the project several times—each time adjusting and fine-tuning it to achieve your original goals. This is called an iterative approach to development.
4. **Save time to polish your project!** Remember to save some time at the end for polishing the final project by, for example, adding special clipart and sounds or music.

Be flexible with the varied pace of your student's abilities and knowledge. Timing will vary and it's important to be patient as they code and give students time to understand how the code, commands, and logic work and not just copy the model.

Let students know that it's okay to change plans once the work has started. Plans are not

something carved in stone. It's a good idea to evaluate a plan at key development points in order to adapt it based on new ideas and skills learned. This project assumes frequent reviews of the plan, as described above, as new features are added.

Students should be reminded that a good way to get an overview of their procedures to better understand the structure is by collapsing procedures by clicking the triangle beside the **to** in each procedure name. This allows students to see the 'forest' and not get lost in the 'trees' and works like an outliner in a word processor. Note: If the **To** is capitalized, it will not work. It must be lower case `to`.

Emphasize to students the importance of testing their procedures at each iterative incremental phase. They should not just implement all procedures and test only at the end. This latter approach makes it much more difficult to find errors of any kind.