LYNX™

LCSI

CREATING A RACE SIMULATION TEACHERS' NOTES

Creating a Race Simulation - Teachers' Notes

Project Overview

This project consists of six 30-minute lessons. It introduces and reinforces some basic coding concepts as well as some more advanced ones, such as creating an "on touch” event handler, parallel processes and multiple turtle control.

Students will use built-in commands and student-created programs as well as interactive control objects (such as buttons), graphics and sound to create a computer race simulation.

While creating this simulation, students explore math concepts such as randomness and its connection to probability.

Prerequisites

Students have already completed the Story - Intro Level for Students

Standards Addressed

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

- Describe the topic, purpose, and audience for media texts they plan to create.
- Construct meaning through the combination of several media “languages” – images, sounds, graphics, and words.
- Plan and write simple computer programs, applying fundamental programming concepts that include clear internal documentation.
- Develop creative solutions for various types of problems that promote talking about mathematics.
- Make connections among mathematical concepts and procedures and relate mathematical ideas to other contexts, daily life, sports, etc.
- Communicate mathematical thinking orally and visually.

Creating a Race Simulation - Teachers' Notes

Before You Start

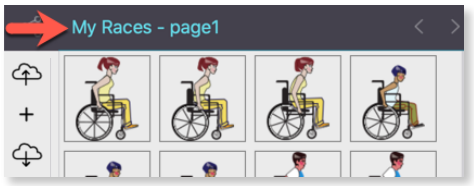
1. At this point your students should already:
   - Have their own Individual account set up on lynxcoding.club.
   - Be rostered in your school's specific club.
   - Completed the Level 1 lessons and created their own story project.
2. Look at the *On Your Mark, Get Set, GO!* sample, at lynxcoding.club, with your students to understand how it works and how it was created. Start by reviewing the following skills:
   - How to use clipart to add a background.
   - How to add turtles (sprites).
   - How to add shapes for the turtle.
   - How to describe turtle movements using commands.

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

- How can you get each turtle to move back to its pre-determined start position?
- How is each race started?
- Can the same instructions describe the movement of different racers?
- How does a racer know if it has reached the finish line?
- What happens once a racer wins? What other reactions can be created?

Have a conversation with your students about the importance of planning. Suggest students write an outline, create a planning diagram, or use another planning tool to help them prepare.

RENAME and **SAVE**: Have your students immediately rename and save their projects with a new name by clicking on "Races Template - page1" above the Procedures Pane and typing in a new name — e.g., **My Races**



Click Save (Arrow to Cloud icon). Lynx creates their own copy of the "Races Template" project in their personal area with its new name — e.g., **My Races**

It is important to remember that Lynx ***does not automatically save their work***, so they must remember to ***save your projects often***!

Creating a Race Simulation - Teachers' Notes

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
	Use basic coding commands to move a turtle (sprite) and create simple animation.
	Use included and imported clipart to create a background, stamp images, and change turtle shapes while learning about image screen layers.
	Write, edit, and debug simple user-defined programs.
	Use commands to create more complex animations and explore multiprocessing.
	Add interactive control with buttons and programmable sprites.
	Add additional multimedia features, such as sound effects and music.
	Use the On Touch event handler to trigger specific actions under specific conditions (in this example, when a racer reaches the finish line).
	Use the word data type to create a personalized response to an event.
	Use screen coordinates to set objects (in this case, turtles) to specific places on the screen.

New Concepts and New Lynx Commands

1. Preparing the background
   `setcolour, setc`
   `fill`
2. Working with multiple turtles
   `clone`
   `ask`
   `everyone`
3. Randomness
   `random`
4. Parallel processes
   `forever`
   `clickon`
   `clickoff`
   `stopall`
5. Collision detection
   `on touch event handler`
6. Using basic data types: words
   `word`
   `announce`
   `say`
7. Reset procedure, buttons and UI elements
8. Cartesian coordinates
   `setx`
   `sety`
   `pos`

Creating a Race Simulation - Teachers' Notes

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. Most errors at this level will be the first type.

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.
- 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.

Some Specific Notes

Additional questions to discuss:

- Can you use the same racing procedure for all competitors or will that make the race results more predictable? Will the random values create equivalent results over a long period of time?
- Why is it important to give turtles meaningful names? Different things can happen as soon as one racer reaches the finish line, for example the race stops and the winner's name is announced, or the race continues until all the racers finish, etc. If students want to announce the winner, the announcement must be unique. Just announcing "I won!" is not helpful if several racers reach the finish almost simultaneously.
- The finish procedure uses `announce` and `say`.

`Announce` and `say` each use a single input that, in this project, is a text string or word, which is what a text string is called in Lynx. There are a few ways to create or indicate a word:

- Use a single quotation mark (`'`) at the beginning and end of a text string: `'hello, world'` is a text string (or, in Lynx, a word) with 12 elements (in this case, the sixth element is a comma, the seventh a space). For example, type in the Command Centre:

```
show 'hello, world'
hello, world
```

- Use a double quotation mark (`"`) at the beginning of a text string. This is used when a text string doesn't contain special symbols (space, etc.): "Elena is a text string (word) containing 5 letters (word).

```
show "Elena
Elena
```

The main commands that operate with words are `word`, `count`, `item`, `word?` . In the sample project, the `finish_line_touch` procedure uses the command `word` which creates one word by combining two or more inputs. So:

```
word 'HowDoYou' "Do?
```

… reports the word `HowDoYouDo?` to a command (such as `show` or `print`) that uses a text string as input.

For example, in the Command Centre, type:

```
show word 'HowDoYou' "Do?
HowDoYouDo?
```

- Encourage students to brainstorm on how they may want to further enhance their projects.

Final Notes

Each lesson should take about 30-45 minutes (although, if students have more time and want to continue, there's lots to explore!).

To save time looking for clipart, there is a Races Template that contains a few pre-made shapes of people and animals moving from left to right and a finish line. Find it inside the All Projects section at lynxcoding.club then look inside the Templates folder

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

On the other hand, 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.