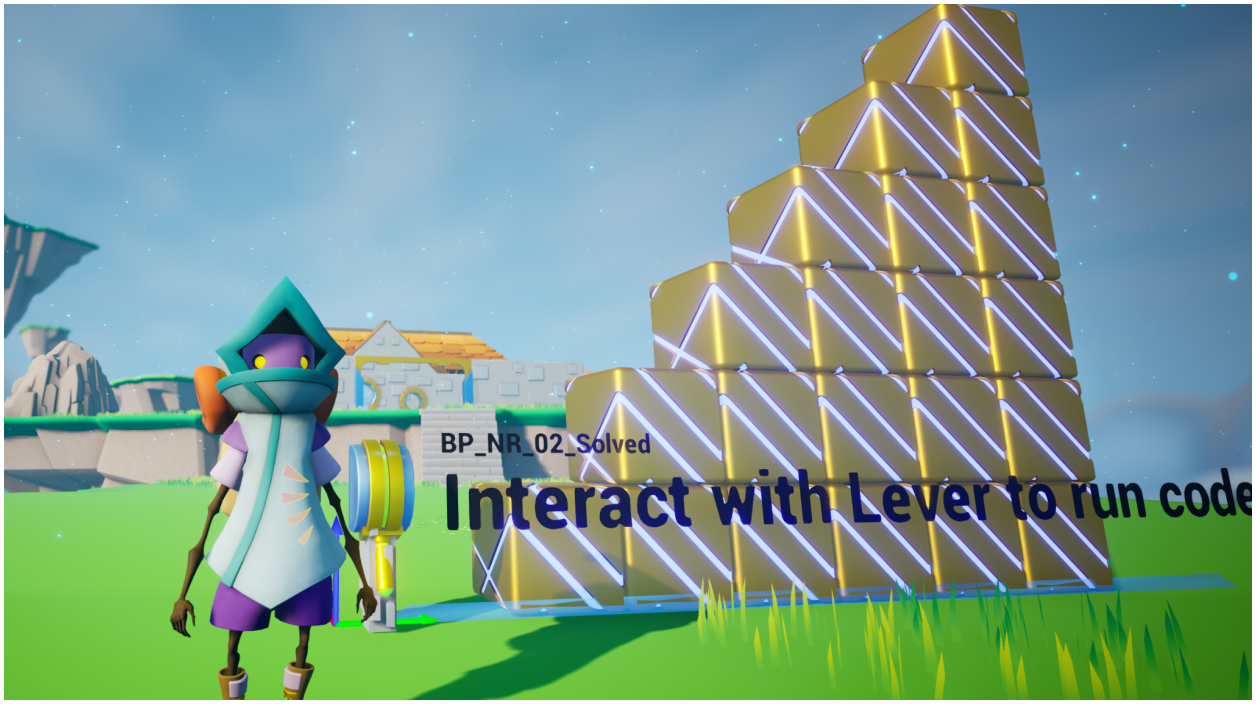


Unreal Engine - Interactive Puzzles Instructor Guide

Guiding Principles

Interactive Programming Puzzles are designed to engage the learner with a hands-on learning activity in which they observe a completed task, and are then given the scaffolding needed to complete the task themselves.

The overarching principle of presenting problems as puzzles with the pieces pre-selected is to avoid cognitive overload; students can concentrate on the subset of nodes presented rather than the thousands of possible nodes available. Additionally, using Blueprints avoids having to concentrate on the syntax and format required in “written” computer languages.

Additionally, the puzzles are designed to encourage play and experimentation with code and solutions. Students can create their own puzzles (a flipped assignment would be a good assessment piece for this material). Students are also encouraged to experiment and demonstrate their learning at the end of each set of problems.

It should be noted that these puzzles and the educational theory behind them are based on what are known as “Parsons Problems”. Please see the end of this document for references to papers on Parsons Problems.

Breaking things

The fact that students are working in a real development environment does leave some room for them to make changes that would “break” the puzzles. This is to be expected. Students should be encouraged to explore the provided code and make changes. Since they are working with a copy of the initial problem, they can always copy the original problem set and start over.

Classroom setup

Students should have a computer capable of running Unreal. They should receive the entire “Computer Science” project. Teachers may optionally remove the Solved puzzles prior to distribution. It should be noted that no effort will be made to hide the Solved puzzles or limit their distribution so teachers should assume that students may find them easily online.

Once it’s been decided on whether or not to include the solutions, teachers should distribute the project to students in a manner that makes sense for their installation.

How to use these assignments

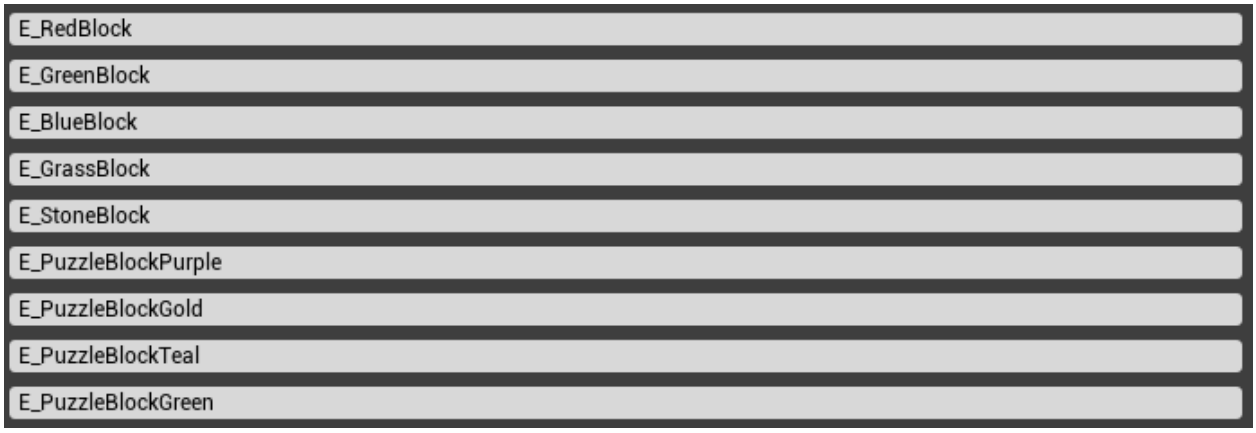
Teachers can create assignments from each of the included documents, use shared journals, or a website to collect and assess student work. Students should be supported throughout their time working on these problems as they may initially need direct instruction on how to access and run the problem sets.

TMI/tl;dr

Below you’ll find some “good to know”, but not necessary, information on how to use these puzzles in a classroom environment. It’s included here for completeness in case someone wants to understand how the project works or create their own puzzles.

Blocks

Most of the puzzles use “Blocks” as a prop. The blocks are 1m cubes (100 Unreal units) and feature various materials. The included blocks are enumerated in the Enum type “E_BlockTypes”. Here are the values:



Puzzles

All Interactive Puzzles have the BP_PuzzleParent as the Parent class. The BP_PuzzleParent takes care of “running” the puzzle as well as placing blocks. The included Lever fires off the “RunCode” node in the child classes, which in turn calls the defined Custom Event.

PlaceBlockOfType

Included in the BP_PuzzleParent is the PlaceBlockOfType function. This function accepts two parameters;

1. RelativeLocation:Vector - this is the X/Y/Z location of the block relative to the point of origin of the puzzle.
2. BlockType:E_BlockTypes - this is the type of block (see above) to place down.

The function stores the new blocks in an array.

When the “RunCode” node in the child returns, the BP_PuzzleParent then calls the FinishAndShowSolution Custom Event, which animates the creation of the solution.

Using an existing puzzle

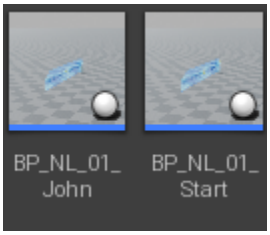
The currently available puzzle categories are:

- Variables
- Single Loops
- Nested Loops

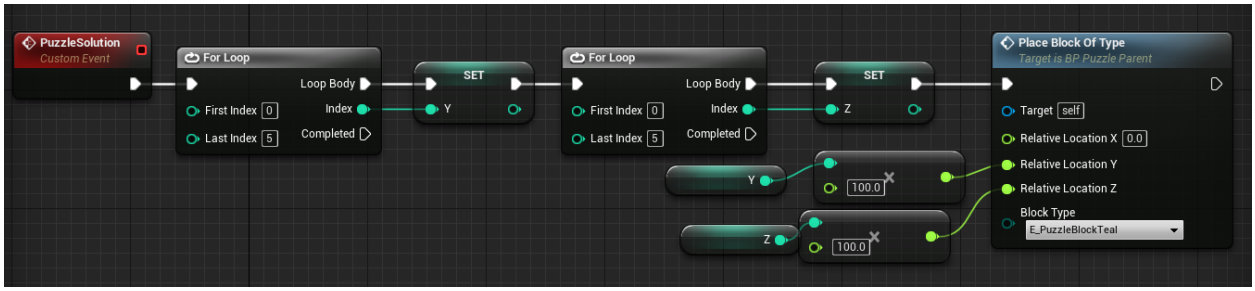
Using the NestedLoops category as an example. To use an existing puzzle, the student would:

1. Open the LessonBlueprints/NestedLoops folder in the Content Browser.

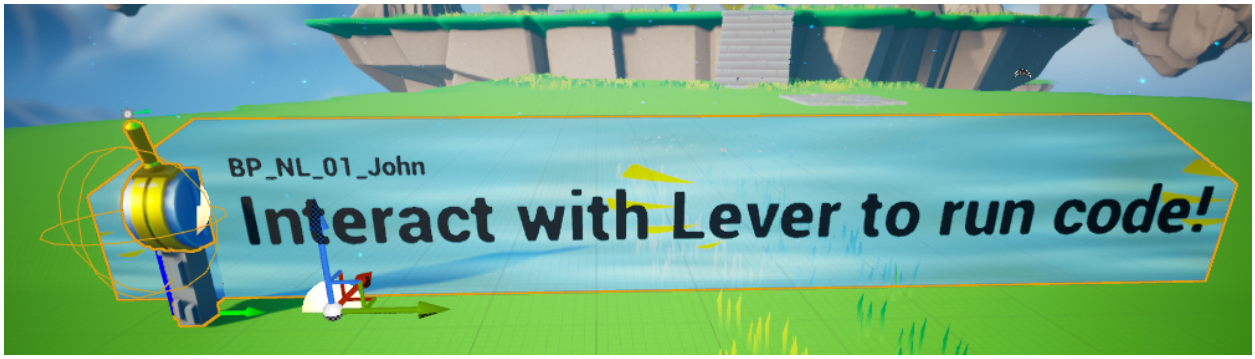
2. Duplicate the Start Blueprint of the problem they are working on.
    a. Example: Student is working on BP_NL_01:
        i. Duplicate (Ctrl+w) BP_NL_01_Start.
        ii. Change the name of the Blueprint to BP_NL_01_Firstname.



3. Open the copy of the Blueprint and find the “Puzzle Solution” Custom Event.
4. Watch the video demonstrating the desired outcome.
5. Arrange/connect the nodes.



6. Place an instance of the Blueprint in the world.



7. Play the game and interact with the lever (left click) to run the code.

8. Observe the output and determine if they wrote the code correctly. Modify/test as needed.

9. Answer the multiple-choice question about the puzzle, which asks the student to identify the correct code.

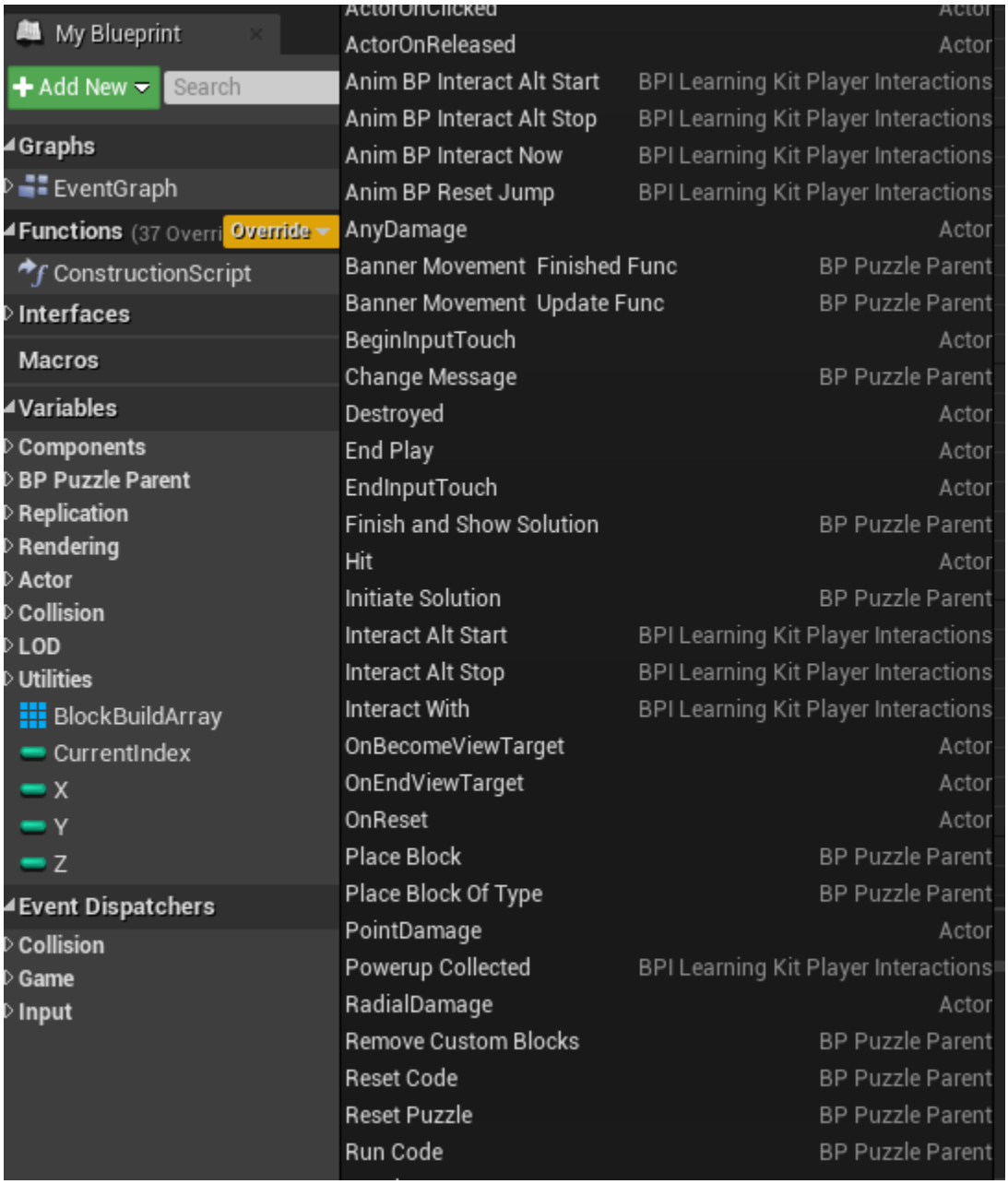
Creating a new puzzle

The easiest way to create a new puzzle is to duplicate an existing puzzle and modify it. There is a skeleton puzzle in the InteractivePuzzle_Content/Blueprints/Solved folder in the Content Browser that can be duplicated to create a new puzzle.

With that said, creating a new puzzle is possible.

First create the “Solved” version of the puzzle:

1. Find the BP_PuzzleParent in the InteractivePuzzle_Content/Blueprints folder.
2. Right-click on it and select “Create Child Blueprint Class” and name the new Blueprint appropriately. Example: BP_NL_01_Solved (BP NestedLoop_NN Solved)
3. Open the Blueprint and Override the “RunCode” function:



4. The RunCode function is called when the puzzle is interacted with (i.e. the lever is pulled down). You have two options:
    a. Write the solution in the RunCode function and instruct students to construct their solutions in that function.
    b. Create a custom event in the Event Graph called “PuzzleSolution” (name is up to you). Call the custom event from the RunCode function. This is probably easier

for students as they will find it easier to access the Event Graph than remember to open the RunCode function.

5. There are three variables available to use from the parent class, X, Y, Z. The two additional variables (BlockBuildArray and CurrentIndex) are read only and shouldn't be accessed in the solution code.
6. Blocks are added with the PlaceBlockOfType function (from the parent class). It has two parameters:
    a. RelativeLocation (Vector): The location to place the block, relative to the puzzle's point of origin. If you are placing blocks, it should be noted that the blocks provided are 1m cubes (i.e. 100 units).
    b. BlockType (enum E_BlockTypes): The type of block to place.
7. The parent class will automatically remove any blocks when the puzzle is reset. If the puzzle you're creating has any cleanup to do, it should override the ResetPuzzle function.
8. Once the solution code is created and debugged, a short video clip of the code in action should be recorded and made available to students. The "Unreal Engine - Interactive Puzzles: Nested Loops" document is an example and suggested format for using Interactive Puzzles.

References

Evaluating the Effectiveness of Parsons Problems for Block-based Programming | Proceedings of the 2019 ACM Conference on International Computing Education Research

Solving parsons problems versus fixing and writing code | Proceedings of the 17th Koli Calling International Conference on Computing Education Research

A Review of Research on Parsons Problems | Proceedings of the Twenty-Second Australasian Computing Education Conference

Editing history

V0.01 2021--6-28 - Initial Release