## Learning hook

* Students first play the simple game against the teacher.
* Students then play the more sophisticated version of the game against the teacher.

## Learning map and outcomes

Students learn to design an algorithm to solve a problem and describe algorithms using flowcharts and structured English; for example, START, END, IF and UNTIL.  
They trace algorithms for a given input, using desk checking, and develop test cases that correspond to the requirements of the specifications; for example, validating program behaviour on a range of valid and invalid user input.

## Learning input

**Building the basic Hi-Lo game in Scratch**

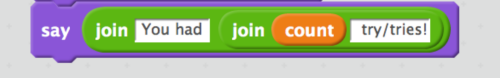
Some people who are strong visual thinkers prefer to use flow charts to develop the basis for an algorithm.

1. Introduce both flowcharts and structured English.
2. Teach the basic control structures both in flowchart and structured English forms.
3. Issue the '[Hi Lo flowchart worksheet 1](https://www.digitaltechnologieshub.edu.au/docs/default-source/getting-started-years-9-10/getting-warmer/hi-lo-flowchart-worksheet-1.pdf?sfvrsn=0)'.
4. Students cut up the symbols on their '[Hi Lo flowchart worksheet 1](https://www.digitaltechnologieshub.edu.au/docs/default-source/getting-started-years-9-10/getting-warmer/hi-lo-flowchart-worksheet-1.pdf?sfvrsn=0)', and arrange them on a piece of paper to form a logically correct flowchart. They draw flow lines where required and add the True/False labels.
5. Initial each correct flowchart. Ask those students to construct an equivalent version using structured English.
6. Explain that structured English is very similar to pseudocode, but it tends not to use as many mathematical symbols.

Often, people start with structured English and/or a flowchart, convert it to pseudocode and then write executable code.   
Explain that structured English avoids mathematical symbols and uses words wherever possible.  
See Teacher Introduction for more information.

1. Demonstrate the basics of Scratch.
2. Ask what differences exist between scripts in structured English and Scratch.
3. Assist students to locate each of these standard programming features, which all appear in the Scratch program the students will develop.
   * Algorithm (whole program)
   * Input
   * Output
   * Sequence
   * Selection
   * Repetition
   * Variable
   * Initializing a variable
   * Operators (arithmetic and relational)
   * Concatenation
4. Assist students to use Scratch to convert their structured English into a working Hi-Lo program.
5. Assist students modify their programs to include a counter that reports the number of tries.

This will require the use of two 'Join' operators:



**Solution**



## Learning construction

**Building the more sophisticated Hi-Lo game in Scratch**

1. Discuss with the class what changes would need to be made to their earlier algorithm to play the second more sophisticated version of this game, using the terms: 'boiling' ,'hot', 'warm', 'cold' or 'freezing', with these rules:
   * boiling: within 5
   * hot: within 10
   * warm: within 20
   * cold: within 40
   * freezing: greater than 40.

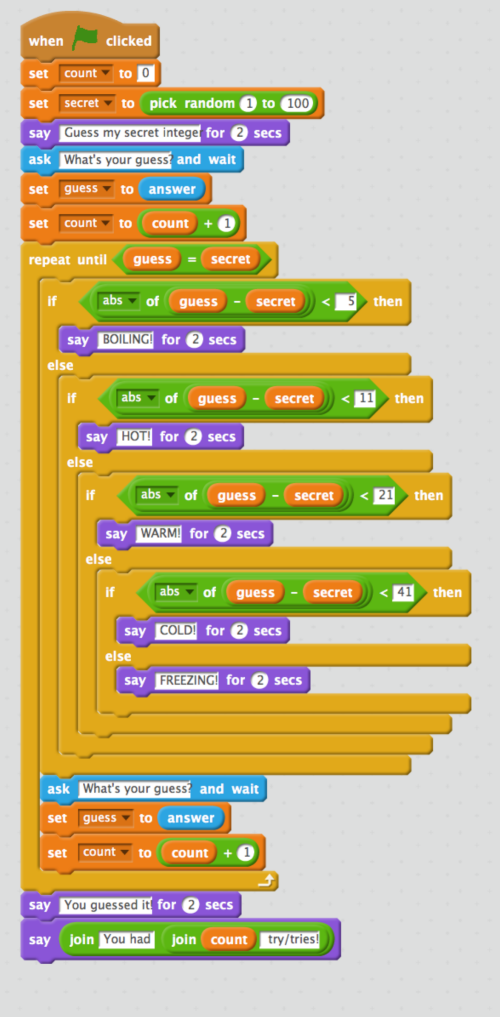
Note: Structured English is easier to use when many choices exist, as is the case here.

1. Having established this, ask if flowcharts or structured English would be more helpful in developing the new algorithm.
2. Students cut up the symbols on their **'**[Hi Lo flowchart worksheet 2](https://www.digitaltechnologieshub.edu.au/docs/default-source/getting-started-years-9-10/getting-warmer/hi-lo-flowchart-worksheet-2.pdf?sfvrsn=0)**'** . They arrange them on a piece of paper to form a logically correct flowchart. They draw flow lines where required and add the True/False labels.
3. Students modify their written algorithm using structured English to create the more sophisticated version of this game using the terms: 'boiling' ,'hot', 'warm', 'cold' or 'freezing'.
4. Students now modify their Scratch program using their new structured English algorithm to play this latter version. This is not a trivial exercise as there are pitfalls that will illustrate well the necessity for desk checking.

Here are some common errors and challenges:

* Students need to find a method of testing if a guess is within a range of the secret integer. However, they encounter negative integers and must discover the ABSOLUTE VALUE function, which is provided in Scratch as one of many functions on the dropdown attached to the SQRT function.
* Students put the conditions in reverse order and thus guesses are immediately satisfied by the first test.
* Students use nested IF ... THEN instead of nested IF ... THEN ... ELSE.
* Students neglect to reset GUESS and to increment COUNT*inside* the repeat loop.
* Boundary conditions can be challenging to get right (for example, a guess 'within 20' will be satisfied by ABS(guess-secret)=20. There is no*less than*or*equal to* in Scratch, so students must use another approach, such as employing the OR statement or using <21 instead.

**Solution**



## Learning demo

1. Explain and demonstrate desk checking.
2. Class discussion establishes suitable test data for this program by forcing 65 to be the secret number.

(Students to test each of: following:  
within each range, boundaries, equal to secret number, decimal entries, negative integers, out-of range integers, invalid entries such as words, null entry, etc)

1. Students prepare a set of values to desk check, including invalid data types and out-of-range values (eg negative integers, integers greater than 100).
2. Students modify their programs to force the SECRET integer to be, say, 65 to enable all students to create comparable test cases.
3. Students desk check each other's programs using a table, and mark as either pass or fail.

## Learning reflection

1. Ask for suggestions of another real-life game that could be programmed (eg Rock/Paper/Scissors, Nim or Noughts and Crosses).
2. Students discuss whether they found the flowchart tool or the pseudocode/structured English more helpful.

## Curriculum links

| Links with Digital Technologies Curriculum Area | |
| --- | --- |
| **Strand** | **Content Description** |
| **Processes and Production Skills** | Design algorithms involving logical operators and represent them as flowcharts and pseudocode [(AC9TDI10P05)](https://v9.australiancurriculum.edu.au/f-10-curriculum.html/learning-areas/digital-technologies/year-9_year-10/content-description?subject-identifier=TECTDIY910&content-description-code=AC9TDI10P05&detailed-content-descriptions=0&hide-ccp=0&hide-gc=0&side-by-side=1&strands-start-index=0&subjects-start-index=0&view=quick).  Validate algorithms and programs by comparing their output against a range of test cases [(AC9TDI10P06)](https://v9.australiancurriculum.edu.au/f-10-curriculum.html/learning-areas/digital-technologies/year-9_year-10/content-description?subject-identifier=TECTDIY910&content-description-code=AC9TDI10P06&detailed-content-descriptions=0&hide-ccp=0&hide-gc=0&side-by-side=1&strands-start-index=0&subjects-start-index=0&view=quick). |

## Assessment

Note: Criteria are cumulative

|  | **Quantity of knowledge** | | | **Quality of understanding** |
| --- | --- | --- | --- | --- |
| **Design and validate algorithms** | No evidence of understanding | Student is able to build a program using Scratch to play some version of the game | Student is able to build a working program using Scratch to play a version of the game with more than three responses | Student is able to build a working program using Scratch to play a full version of the game and with the option of playing again | Student is able to build a working program using Scratch to play a fully tested version of the game, with error messages for invalid entries by users |
| **Optional score** | 0 | 1 | 2 | 3 | 4 |