BY NC ND

code-it.co.uk

My Best Game Ever

Pupils learn about three forms of conditional selection, selection starts action, selection switches between actions, selection stops repeated action. They write their own everyday examples of these forms of conditional selection before learning about how two of these can be adapted by the addition of a forever loop to tests the conditions continually. They do this through writing their own gaming examples during roleplay on the playground. Finally they think through a gaming plan of their own before incrementally planning a part of the algorithm, programming it and testing it.

Computer Science Concepts

Computational thinking

Decomposition

(breaking up a project into manageable chucks and solving them separately)

Algorithmic thinking

-Sequence –Repetition

–Conditional selection

National Curriculum Programs of Study

Pupils should be taught to:

design, write and debug programs that accomplish specific goals, including controlling or simulating physical systems; **solve problems by decomposing them into smaller parts**

use sequence, selection, and repetition in programs; work with variables **and various forms of input and output**

use logical reasoning to explain how some simple algorithms work and to detect and correct errors in algorithms and programs

Plan

1, Learn about conditional selection

2, Forever loops to continually check the condition

3, Roleplay conditions on the playground using shapes

4, Plan a big idea

5, Repeat until the game is complete.

Plan a small part of the algorithm,

program,

test it.

6, Evaluate the game

Scratch 2.0 or 3.0 Planning by Phil Bagge

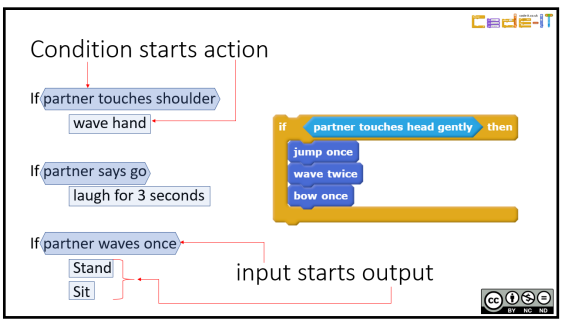
Code-it.co.uk

1, Learn about conditional selection

Use the PowerPoint provided to learn about conditional selection. Pupils work in pairs with a whiteboard and a pen or other writing implements. Note the use of non scratch language for loops this is to encourage pupils to think of the design level as separate from code.

Slide 2 Pupils will take it in turns to be the human input for the algorithms or the output and act out the program.

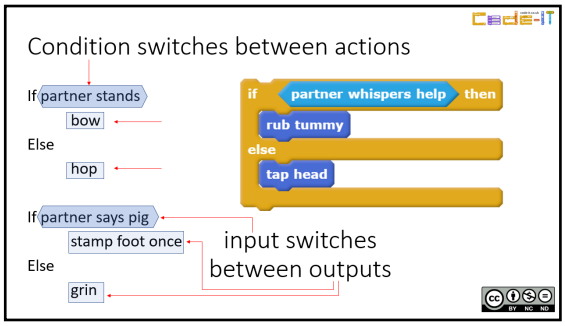
Slide 3 Pupils are introduced to condition starts action. Ask pupils to identify which part is the condition and which is the action. At the end of the slide get every pupil to write an everyday algorithm where a human input starts a human output. Their partners can test these. Award 1 mark for a condition that starts an action or actions. Award two marks if pupils clearly separate the condition from the action. Award one mark if pupils indent. Don't ask pupils to rewrite just make a fuss of those who get all three points and suggest others can improve by aiming for



Slide 3

all three points. The most important thing is that it makes sense. The other aspects help pupils to organise their thinking.

Slide 4 Pupils are introduced to condition switches between actions. Ask pupils to identify which part is the condition and which are the actions. Ask them what happens if the condition is not met? Answer the else part of the algorithm is enacted. At the end of the slide get every pupil to write an everyday algorithm where a human input switches between two actions. Like before, get pupils to test these on each other and use the three points system to encourage accuracy.



Slide 3

Slide 5 Pupils are introduced to condition stops a repeated action. Ask pupils to identify which part is the condition and which are the repeated actions. Ask them what happens if the condition is not met? Answer the repeated actions will continue until the condition is met. At the end of the slide get every pupil to write an everyday algorithm where a human stops a repeated actions. Like before, get pupils to test these on each other and use the three points system to encourage accuracy and well laid out algorithm.

2, Always loops to continually check the condition

Slides 6 and 7 Introduce the idea of putting a condition inside a loop so it is enacted over and over again. There are a couple of examples for pupils to act out on slides 6 and 7.

3, Roleplay conditions on the playground using shapes

Slides 8 and 9 introduce a role play example of this, where pupils act out conditions inside forever loops before writing and testing their own examples. The cards to act out and the templates to write their own are available to print out as PDFs. You will need to chalk some shapes on the playground or draw some using masking tape on the classroom floor before using this activity. This really helps pupils to think and plan in the next section.

Algorithmic thinking

Refers to the use of logical concepts such as sequence, repetition, selection, variable use, functions etc to design an algorithm.

Algorithm

A logical order of instructions to complete a task or solve a problem.

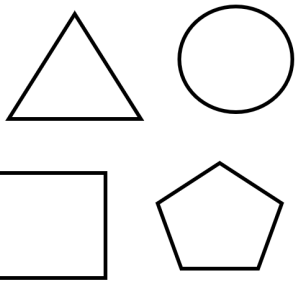
Algorithm and code

If a set of instructions written out is exactly the same as the code then it is code.

Common misconceptions

Many pupils miss out the continuous loop in algorithms and code.

When you demonstrate a condition in a loop continually ask if the condition has been met. This helps pupils to comprehend how the loop is interacting with the condition.



4 levels of abstraction

Task level for brief big idea

Design level for

List of objects / characters

What you want characters to do

Initialisation (Making game run the same every time)

Algorithm (detailed steps to turn what you want characters to do into algorithms)

Code level

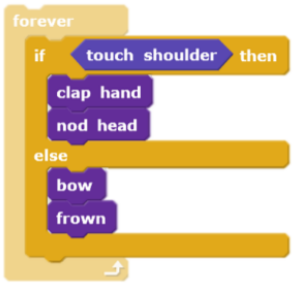
Turning algorithms into code

Execution or running code level

Testing code

Creativity

The choices around what game to make and what it will look like are totally open to every pupil however teachers can be assured that they have moved pupils programming forward by teaching the concept of conditional selection carefully and without overloading pupils memory



Roleplay activity

4, Plan your game

Slide 10 introduces pupils to writing a task level for their game. This should be a brief couple of sentences.

I want my cat to move and steer round a maze towards the finish trying to avoid slowly moving dogs.

Slide 11 introduces objects and what they will do. Focus pupils on thinking about one character first.

Slide 12 introduces how to take one thing that we want our characters to do and turn it into an algorithm before coding. Remind pupils of the examples on second page of the planner.

5, Repeat until the game is complete

Slide 11 introduces the idea of planning a part of the algorithm before programming it testing it and then repeating this process.

If pupils are looking blankly you could say who has a sprite that will move when a button is pressed? Say they will not be allowed to start coding this until they have written an algorithm for it first and checked it with you. Stress the cyclical nature of the process. Once pupils have completed a couple of these cycles they start to get the hang of it. You could also model one first. *(there are examples of pupils work on the next page)*

6, Evaluation

Whilst an end of game evaluation can be useful for summative assessment I would recommend a review 3/4 of the way through the build so pupils have time to act on peer and teacher recommendations for improvement. Asking pupils what make a good one of these WAGGOL is a great way of collecting criteria to assess pupils projects.

Another way of assessing pupils ability to create great algorithms that include selection is to ask them to plan something that will be programmed in another language. I use Crumble for this.

Waking up Pupil Computing Brains

I have also included a roleplay and questions activity to use as lesson starters to help wake up pupils computing brains and remind them how conditional selection works.

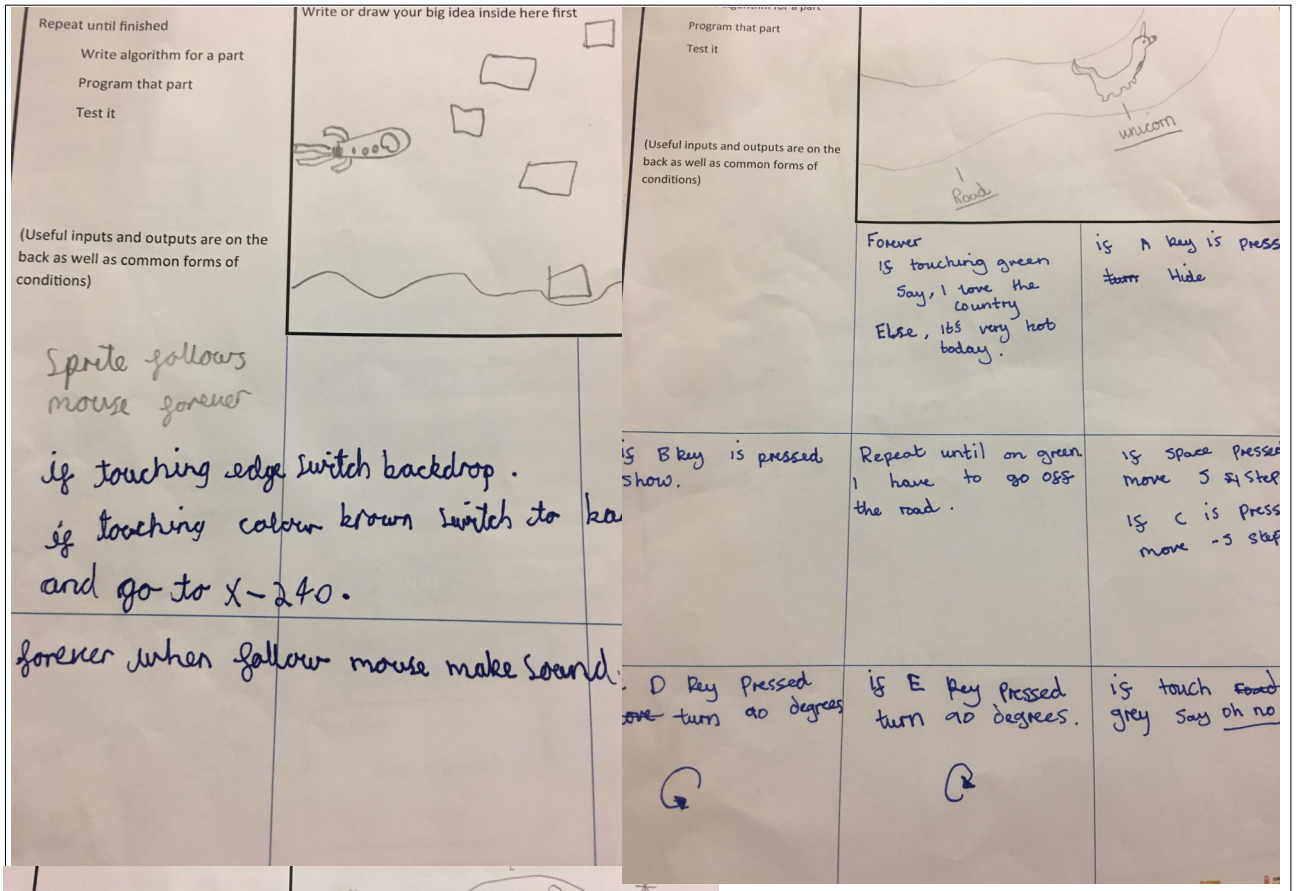
Final Word

I hope you enjoy this planning as much as I have done. If you use the module do email and say how you get on positive and negatives comments are welcome as it helps me to improve it.

Phil Bagge

29th October 2018 Revised 25th October 2019

The last page has useful problem solving skills you may wish to promote throughout the project design and build.



Helping pupils think about mistakes is often easier in the algorithm stage.

Have they only checked the condition once? Add an always loop.

The examples were created before I realised the importance of using non Scratch language to develop their algorithmic thinking.

There is clear evidence from Armoni[1] et al that novice programmers need this layer to be in language that is similar but not the same as the programming languages used as it helps them to think algorithmically rather than in code. This makes it easier to transfer from one programming language to another or to create algorithms which are not code related. Waite[2], writing in 2017, observed that teachers tend to confuse the algorithm and code levels.

1 Teaching abstract thinking in Computer Science for 7th graders Armoni et al (2013)

2 Comparing K-5 teachers' reported use of design in teaching programming and planning in teaching writing Waite (2018)

Computing Problem Solver

I recognise there is more than one way to solve/describe a problem

I don't just accept the first solution

I look for a range of solutions to the same problem

I look for how a project can be extended

Handles Ambiguity

Open Ended Problem Solver

I can break complex problems into parts

I can evaluate my solutions against a set criteria

I can design criteria to evaluate my creations

Evaluates

Copes with Complexity

I can discover/concentrate on the most important part of a problem

I can identify patterns in problems and solutions

I can contribute useful ideas to a partner or group

I can encourage others to share their ideas

Communicates

Adapts

I can adapt existing ideas to solve new problems

I can develop, test and debug until a product is refined

I lead using all the people talent in my group

I learn from setbacks and don't let them put me off

I can persevere even if the solution is not obvious

Perseveres

Investigates

I make predictions about what will happen

I repeatedly experiment through predicting, making, testing & debugging

CC BY NC ND

Adapted from a problem solving rubric created by Mark Dorling & Thomas Stephens hosted at http://code-it.co.uk/attitudes/