Queen Mary
University of London

Spit-Not-So

Age group: 10 – adult

Abilities assumed: Nothing

Time: 20-30 minutes

Size of group: 1 upwards

Focus

Computational Thinking: translating problems & understanding people
Human-computer interaction
Graphical User Interfaces versus command line interfaces
Data structures

Syllabus Links

This activity can be used as a general introduction to human computer interaction from KS2 upwards and in particular as an illustration of why graphical user interfaces are often easier to use than command line interfaces. It can be used to introduce the idea of a data structure and show how data structures choice makes a difference to how well a task can be done. It also introduces the computational thinking ideas of translating problems and understanding people.

Summary

You play a simple word game with the class. You show that "due to you computational thinking skills" you are brilliant at it – and never lose. You eventually show the class the secret, and in doing so illustrate a reason why graphical user interfaces are easier to use than command line interfaces and also why the choice of data structure is so important. Oh, and you also shed light on what computational thinking is all about.

Technical Terms

Algorithm, computational thinking: translating problems, interaction design, human-computer interaction, graphical user interface (GUI), command line interface, data structures.

Materials

Board, flip chart or OHP
Board markers of two different colours
Spit-not-so cheat sheet (laminated)
List of Spit-not-so words
Perfect algorithm for playing Noughts and Crosses sheet (laminated)
Pen and pad

What to do

The Grab:
Explain you are going to teach everyone a simple pencil and paper game called Spit-not-so[1]. You are brilliant at playing it because of your computational thinking powers: the skill you learn from studying computing. You will show them how to be brilliant at it too!

The activity:

Write the nine Spit-not-so words on the board in a vertical list with space between:

SPIT

NOT

SO

FAT

FOP

AS

IF

IN

PAN

Next explain the rules as follows. The game is played by two players, one playing red and the other blue. Red goes first. Each player takes it in turns to claim a word. They do this by circling the word in their colour. Once a word has been claimed, the other player can no longer claim it.

To win, a player must claim three words that contain the same letter. For example if you had claimed SPIT, IF and IN, you would have all three words containing the letter I so would win. Once all the words are claimed, if no one has won, then the game is a draw. Players are allowed pencil and paper to make notes in any way that will help them play the game.

A game might go:

PLAYER 1: SPIT
PLAYER 2: SO
PLAYER 1: FAT
PLAYER 2: NOT
PLAYER 1: FOP
PLAYER 2: IF
PLAYER 1: PAN

At this point Player 1 wins as they have the cards: SPIT, FOP and PAN – three Ps.

Get the class to play a few games in pairs to get the idea. Then ask them about the

[1] The game of Spit-not-so along with how to win is described in E. R. Berlekamp, J. H. Conway and R. K. Guy (1982). *Winning Ways for your Mathematical Plays*. Academic Press.

game:

- Was it easy to tell when someone had won?
- How many games ended in wins and how many in draws?
- How hard did they find it to play the game?
- Did anyone feel they made a silly mistake – missing an obvious word they should have taken?
- How easy is it to see when the other player is going to win?
- How easy is it to see good and bad moves?
- Take a vote: compared to a game like noughts and crosses, say, (which you assume everyone knows) is it easier or harder to play Spit-not-so well?

Next challenge the class to a game against you. Have a volunteer come to the front to play. The rest of the class can shout out help. Give the volunteer a pen and pad so they can write notes if they want to.

You should have as your notes the following grid (keeping it hidden).

NOT	IN	PAN
SO	SPIT	AS
FOP	IF	FAT

The position of each word is important. They have been organised so that there is a single letter in common in each row, column and diagonal. All the F-words run along the bottom, for example.

To play the game each time you claim a word, put a cross in the square that corresponds to that word. Similarly put a nought in any square corresponding to a word your opponent claims. For example, the above game will end with your grid looking like the following:

NOT	IN	~~PAN~~
SO	~~SPIT~~	AS
~~FOP~~	IF	~~FAT~~

While your opponent is playing the tricky game of *Spit Not So*, you are just playing *Noughts and Crosses*. To be extra sure of winning, use our perfect noughts and crosses algorithms sheet to tell you where to go.

Once you have played a game or two, ask the class if they know how you can do it so easily. Can they work out what you are writing as your notes? How could you make the game easier to play? What did you prepare earlier? After their suggestions reveal the secret. While they have been playing Spit-not-so, you have been playing noughts and crosses (and following an algorithm to do it). Talk them through a game with the words now also written in the grid rather than in a line to illustrate.

Point out that the game is not easier or harder to play than noughts and crosses. It is exactly the same! But only if you organize things first!

The explanation:

There are several different lessons that can be derived from this activity. Choose those that fit your lesson aims.

Data structures
Spit-not-so shows that the way we organise information can make a massive difference to how easily a task can be completed using that information. In the basic game, the information is organized as a list. To answer questions about that information such as: "Can any one win next move and how?" is quite complicated: keeping track of all the letters each player holds and which are still free.

By organizing the information into the grid structure before playing we make it easy. We now just look at the lines in the grid, watching especially for ones with two noughts or crosses and a space. In computing terms, we are just using a different data structure – a different way of organizing the same data. By changing the data structure from a list to a grid we are able to change the algorithm needed to achieve the task from a slow one to a fast one. When we write programs, the same issues apply. We can often devise faster algorithms if we come up with a way to organize the data appropriately first. In Spit-not-so it is a human trying to do the computation needed to try to win – following the algorithm – but the issue is the same if it was a computer doing it. A slower algorithm is a slower algorithm whatever or whoever is following it.

User interfaces
Another way of thinking about Spit-not-so is that it is not just about the way the information is organised but also about the way that information is then presented to the player. By presenting it in a grid we made it really easy for a person to process. Instead of needing to process words and letters – using our linguistic skills, it becomes a simple visual pattern-matching job. We do not have to pay any attention to the words at all, we just look for lines. That's fairly simple for us, because we are visual creatures. A large proportion of our brains are dedicated to visual processing. Looking for visual patterns is something human brains are really good at. By contrast processing and counting letters is something that is much harder for us. Not only are we likely to find the noughts and crosses game quicker and easier to play, we are far less likely to make big mistakes like missing a winning move or missing that we should block a winning move.

When writing programs, we have to create a human-computer interface. In part this involves designing the way that the program presents its information to the person using it. The difference between the two ways of playing Spit-not-so is essentially the

difference between a command line interface and a graphical user interface. The former relies on people remembering and typing command words – it is all about language. The latter is all about visual things – finding and recognizing commands from icons and their position, for example. Just like Spit-not-so, graphical user interfaces are easier because they play to our visual strengths.

By choosing to present the words in a grid with each word in a particular position, we turned a game that is quite easy to make mistakes when playing, to one that you are unlikely to lose. That is what choosing a suitable presentation, based on a suitable organisation, together with an appropriate set of rules to follow, can do for you!

There are many situations when you do not want people using computers to make mistakes too: when the computer is controlling a nuclear power plant, is an air traffic control desk, or is a medical device keeping someone alive in a hospital. We must then be sure we design the interface so it helps the people using the computer avoid making mistakes.

Computational thinking

Computational thinking is the general problem solving skill set that is associated with computing. Spit-not-so illustrates several aspects. The first is the computational thinking idea of transforming problems. Rather than solve problems from scratch we can look for similar problems we have already solved and transform the problem. If we transform the solution to fit the new problem we have a ready-made answer. We did this with Spit-not-so. Realising it can be transformed into noughts and crosses, we did not need to work out a winning strategy for the new game. Instead we took our existing strategy for playing noughts and crosses and applied it to the transformed version of the new game: we immediately can play Spit-not-so perfectly.

It also shows how computational thinking solutions are not one-off answers but algorithms. We haven't 'solved' Spit-not-so just because we won once, we have solved it when we can always play it perfectly: when we have an algorithm for winning. In this case we just built that algorithm around one we already had for winning noughts and crosses.

A final aspect of computational thinking Spit-not-so demonstrates is the importance of understanding people when problem solving. By changing they way we present information we are able to make that information far easier for people to process to achieve the particular task that needs doing. Here, that task was keeping track of who has what, for each triple of words with a common letter. Making it easier for people is not only about putting the words in a grid. We could have still marked the moves by circling words in the grid with the colour of the player who had claimed it. Instead we used Os and Xs which made it visually the same as noughts and crosses, helping the person play that game. By thinking about people when coming up with the way we represent the information we can make the game as easy as possible for a human to play. Similarly programs are often there to help people do some task, we can give them an interface that makes it easy for them to work with the program or one that makes it hard. If we want to help them we need to make the program work with their abilities. Computational thinking problem solving has to take the abilities of people into account too.

Variations and Extensions

Creating a cheat sheet
For a longer session, finish it by having the class play some games of Spit-not-so using the noughts and crosses cheat sheet. Ask the questions again at the end about how easy or hard it is. Was it easy to tell when someone had won? Were there more draws playing this way?

Creating a cheat sheet
Have the class try and work out the positions of the words on the grid for themselves in groups once they know the principle but without them having had time to take in exactly where the words go. This is quite hard from scratch, but a hint you can give is to suggest they think about where 'SPIT' must go. They should also think about the number of lines each square is on, and the number of letters in the words. If there are three letters in the word then that word must be placed in a square on three winning lines (a corner).

Invent a new version of the same game
Another (fairly hard) task is to try and invent a new version of the game – can they come up with a different set of words – perhaps including longer words with letters that can't be used in a triple. Can they come up with a different but equivalent game? For example, what happens if you make the board a map with the squares of noughts and crosses cities connected by roads?

Further Reading

Computing without computers

A free booklet by Paul Curzon on programming, data structures and algorithms explained using links to everyday concepts. Available from http://teachinglondoncomputing.org/resources/

Queen Mary University of London

Links to other activities

The Four Aces

Teach a trick where the Aces are stolen from a perfect hand without anyone seeing.

You do a magic trick where the audience try to keep track of the Aces. To their surprise the person who had all the Aces turns out to have nothing. You the magician have the perfect hand. How do you steal the Aces with no one noticing? Magicians design systems so everyone makes mistakes, computer scientists have to design them so no one does. This is a memorable way to show that computing is about more than just technology. It is about understanding people too.

The intelligent piece of paper

Take part in a test of intelligence against an intelligent piece of paper!

This is a good introduction to what an algorithms is and how a computer program is just an algorithm. It can also be used to start a discussion on what it would mean for a computer to be intelligent. It can lead on to an unplugged programming activity creating winning instructions.

Live demonstration of this activity

Teaching London Computing give live sessions for teachers demonstrating this and our other activities. See http://teachinglondoncomputing.org/ for details. Videos of some activities are also available or in preparation.

Department for Education

SUPPORTED BY MAYOR OF LONDON

COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE

chi+med
making medical devices safer

EPSRC
Engineering and Physical Sciences Research Council

This activity sheet was created in collaboration with CHI+MED, an EPSRC funded research project involving University College London, Queen Mary University of London, Swansea University and City University on the interaction design of safer medical devices: http://www.chi-med.ac.uk/

Queen Mary
University of London

Spit-Not-So Cheat Sheet

NOT	IN	PAN
SO	SPIT	AS
FOP	IF	FAT

Computer Science activities with a sense of fun: Spit-not-so V1.0 (15 March 2014)
Created by Paul Curzon, Queen Mary, University of London for
Teaching London Computing: http://teachinglondoncomputing.org

Queen Mary
University of London

Algorithm for playing Noughts and Crosses (playing first)

Move 1: Go in a corner.

Move 2:

IF the other player did not go there
THEN go in the opposite corner to move 1
ELSE go in a free corner.

Move 3:

IF there are 2 Xs and a space in a line
THEN go in that space
ELSE IF there are 2 Os and a space in a line
THEN go in that space.
ELSE go in a free corner.

Move 4:

IF there are 2 Xs and a space in a line
THEN go in that space.
ELSE IF there are 2 Os and a space in a line
THEN go in that space.
ELSE go in a free corner.

Move 5: Go in the free space.

Queen Mary
University of London

Algorithm for playing Noughts and Crosses (playing second)

Move 1:

IF the other player went in the middle
THEN go in a corner
ELSE go in the middle

Move 2:

IF there are 2 Xs and a space in a line
THEN go in that space.
ELSE IF there are 2 Xs in opposite corners
THEN go in a free space on the side
ELSE IF the other holds a single corner
THEN go in the opposite corner
ELSE go in a free space.

Move 3:

IF there are 2 Os and a space in a line
THEN go in that space
ELSE IF there are 2 Xs and a space in a line
THEN go in that space
ELSE go in a free space.

Move 4:

IF there are 2 Os and a space in a line
THEN go in that space
ELSE IF there are 2 Xs and a space in a line
THEN go in that space.
ELSE go in a free space.