Project Report for a Painting Program

*How the AI blocks are included into the program:*

This project involves using speech synthesis, speech recognition and pose detection. The idea behind the program is that someone’s hand movement could be traced and copied onto the screen as if they were painting. Speech synthesis is used in the program to give instructions for the game as well as ask the user for information such as which hand they would like to paint with. Speech recognition is used by the user to tell the program instructions. The user is able to communicate verbally what pen colour and size they want to use, as well as telling the program when to start and stop painting. The user’s wrist is tracked using pose detection. This is then replicated by the sprite on screen. When the pen is down, the movement of the wrist is shown on screen. This means that the user can paint by moving their wrist around.

*The experience of using the AI blocks:*

After repeatedly using a say block next to a speak block in the Spelling Game project, I learnt how to create a subroutine which spoke and outputted messages on the screen at the same time. This mean that because I did not have to use many of the speak blocks in the main program, I decided to use the block where you can change the accent to a British accent, which I prefer. The subroutine meant that doing this was neater.

In the spelling game, I used the listen to the microphone and then block, however, it only need to listen at one point in the program. This time it needed to listen out all the time for instructions from the user. I used the new next thing spoken function in an infinite loop to do this. The comment on the block helped me to understand how to use it. However, the user cannot speak answers to which hand they want to use or whether they are happy with the artwork unlike with the theme question in the Spelling Game.

I found that using pose detection was more complicated because I have not used it before. However, I think that what made it more complicated was matching up the pen position with the pose. This is done within the poses block. This meant that it took more time for it to work. For example, with the pose property block, the location is a number that is bigger than 0 so in Snap! to draw across the whole screen, the number had to be enlarged so it included the negative bottom left segment of the graph as well. Furthermore, when developing the program, is seems as if when the pose property of the left wrist is requested, it sends the location of the right wrist. However, when at a close distance from the camera with one arm in the frame, it tracks it even if it is meant to track the other wrist.

*Analysis:*

Below is a basic structure diagram for the program.

There are two sprites used because one has the costume of the pencil, but draws from the centre of the image not the end of the pencil. The other sprite is hidden but does the drawing. The pencil receives the location of the wrist. This is then broadcast to the hidden sprite which will follow the pencil as it moves creating a line from the tip of the pencil. The difference between the centres of both sprites is measured so this works. The pencil is listening out for when the user says ‘finish’ to then stop the loop of locating the wrist. The hidden sprite listens for instructions for stopping, starting, and changing the pen colour and size because this sprite is drawing. For consistency when the user is asked which colour or size they want, the question is said and spoken. However, because the sprite is hidden, the message does not appear on screen. Therefore, the message has to be broadcast to the pencil sprite which will then say the message.

The hand function asks the user which had they want to use. The body part in the pose property bock depends on the return value of this function.

The change function askes the user whether they want to continue editing the painting after they have said ‘finish’. This is because, when they say ‘finish’, the pencil sprite is hidden so the user can view the whole picture without the pencil obstructing it. Then, if they have missed something, they can change it. When the user is happy, they can screen shot the image to save it.

*Variables:*

|  |  |  |
| --- | --- | --- |
| Name: | Type: | Use: |
| pose | array of numbers | Hold locations of all body parts detected by the poses block. |
| more | integer | Holds whether the program will continue to detect the poses. |
| hand | function which returns a string | Holds whether the user is left or right handed. |
| invalid | integer | Holds whether the user’s input is valid or invalid. |
| speech | string | Holds the last thing spoken by the user. |
| text | string | Holds text that is broadcast between the two sprites. |
| wrist | array of numbers | Holds the coordinates of the location of the wrist. |
| change | function which returns a Boolean value | Shoes whether the user wants to continue painting or not. |
| message | string parameter | Holds which message is being broadcast between sprites in the broadcast text block. |
| text | string parameter | Holds the message that is being both said and spoken at the same time by the say and speak block. |

*Validation:*

If the computer does not understand the instructions the user is saying, it will not do anything. If it understands ‘colour’ or ‘size’ but not the colour or size requested then it will output an error message. The program only understands a finite number of different colours, so if unknown colours are requested the same error message will appear. There is validation for the hand function. It accepts ‘l’, ‘left’, ‘r’, and ‘right’ and it ignores capitals and spaces before or after the word. There is no validation for whether the user wants to edit their work because both ‘no’ and ‘n’ are accepted as answers with both capitals and spaces either side. If the answer is random characters or no is spelt incorrectly, the program will stop because it seems the user is not interested anymore.

*Development:*

I changed the scale in Snap! so the drawing filled the whole graph as well as matching which wrists were being used (as discussed above), even though this does not make a difference when close to the camera. Another change I had to make so the picture matches is to swap the y-axis because otherwise, when the wrist moves up, the sprite would draw a line down.

At first there was a problem with broadcasting messages between sprites, however, the problem was that I was using the same name for the parameter and the global variable which holds the text.

After the change function has run with change being true when the user wants to edit, I found that the question kept appearing. This is because speech is still ‘finish’ from when it was finished before so the speech variable needs to be reset during the loop.

*Evaluation:*

I do not think that this is a very useful program because the drawing is not very accurate. However, it is fun to see the computer follow your movements.

*Possible improvements:*

* Change the costume to a brush because this fits with painting, rather than painting with a pencil.
* Edit the ‘hand’ and ‘change’ functions so they use speech recognition and the user can say their answers rather than have to type them.
* Make infinite colours available.
* Ensure that you do not have to repeat instructions or use speech recognition with sentences where, for example, a word is detected along with a number for an instruction.