Criterion C: Product Development

Techniques used

Techniques clearly listed.

Additional information is available in the appendices:

1. Appendix 1 – evidence of development of game
2. Appendix 2 – Bibliography

Links to appendices explicitly made.

Graphical Interface

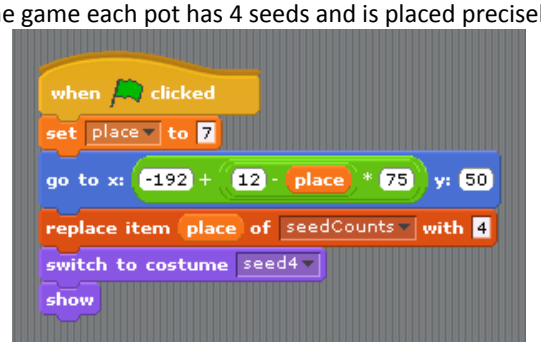
Each pot must have a different aspect according to the number of seeds it contains. This is achieved by giving each pot a set of costumes.

Pot 9 has 4 seeds

pot 2 has 7 seeds

Pots 1 – 6 are chosen by computer 7 – 12 are clicked on by the player

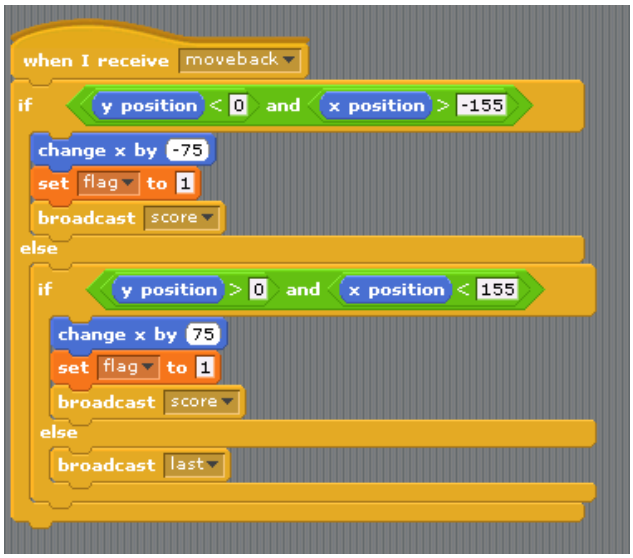
At the start of the game each pot has 4 seeds and is placed precisely on the board.



Pot 7 set on the board with 4 seeds

The number of seeds is stored in the array seedCounts [7]

The precise placing is important as pots change when the sprite is touching the pot. The sprite moves to precise co-ordinates.



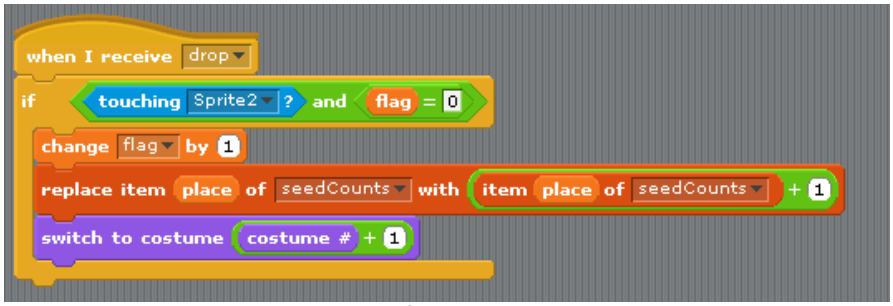
Sprite moves anti-clockwise on the board to next pot and swaps between the two rows if necessary.

Techniques used to develop the game clearly illustrated by screenshots, provides evidence of ingenuity.

page 2

Methods

Methods are called by objects via "*broadcasting*" and messages broadcast are received by everyone. The following code prevents the message being picked up by the wrong pot.



Pot must have sprite touching and the flag ensures that once the sprite has moved on the same message is not used by another pot.

More evidence of complexity and ingenuity in the development of the game.

Variables

local variables
place is local to the pot object and set at the beginning of the game. This is used to set the position of the pot. It also acts as the index in the array **seedCounts**
highest, rank, temp1, temp2 are local to Sprite and are used to calculate the best move
potScores – a **list** local to Sprite used to hold score that each pot could collect if chosen

global variables
computerScore, playerScore used to hold the current scores of players. This are displayed throughout the game
flag, flag2 used to indicate message picked up
seedCounts a global **list** holding the number of seeds in each pot.

The use of local variables for the pots meant that each pot could be derived from the basic pot with little change of coding. The only differences in code are:
- setting the place for each pot
- giving a call for the “move” according to which player the pot belongs to

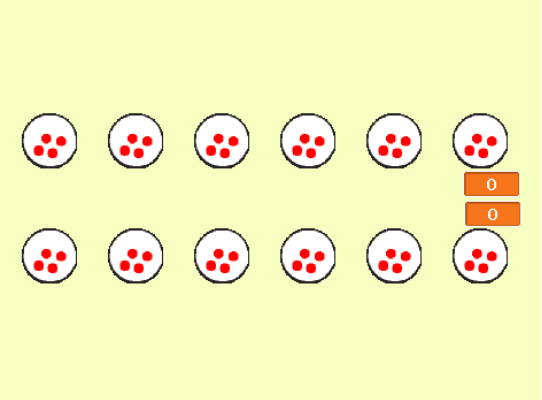
Excellent explanation for use of global and local variables.

Calculating the best move

```
for each pot that belongs to the computer
        set rank to zero
        set temp2 to (number of seeds in pot) mod 12    – calculates the number of moves avoiding
                                                          complete cycles of board
        add pot number and subtract 12                  - finds the pot that will be the final drop
        if pot belongs to player
                while seeds in pot either 1 or 2
                        add number of seeds in pot to potscores [pot]
                        move back                       - calculates the number of seeds that can
                endwhile                                  be taken
        endif
endfor
set highest to 0
set rank to 0
for each pot
        if potscores[pot]  > highest
                set highset to potscores[pot]           - finds pot with highest potential score
                set rank to pot
endfor
if rank = 0 set rank to random number between 1 – 6     - if none score then choose random pot
move sprite to position |rank + 6]
call ComputerGo                                         - start move
```

Evidence of complexity in the algorithmic thinking.

Setting the board was more difficult than expected. At first I placed each pot by hand but this was too inaccurate and as the sprite moved sometimes it was not exactly touching the pot. To get over this each pot moves to an exact position at the start of the game.



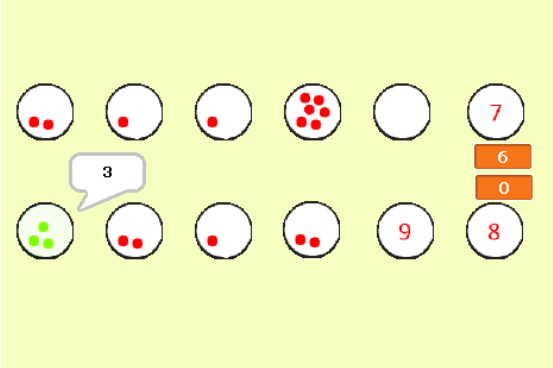
Start of game. Player has the bottom row and computer the top row
Score shown next to row.

page 5

Broadcasting

Another problem was the broadcasting to call methods. There seemed no way to broadcast to a specific object which meant that more than one pot picked up the message. The restriction that the sprite had to be touching was not enough as the sprite moves from pot to pot. Setting a flag once a message had been received stopped this confusion.
When my cousin played we found that there were not enough costumes to each pot and after reaching 12 the pots would go back to zero, giving a wrong picture. This was corrected by 36 costumes for each pot. It seems a bit excessive but sometimes the game did go over 20 in a pot – probably my cousin was doing it on purpose.
Another thing that went wrong was that sometimes, if none of the pots could give a score, the computer chose a pot which had no seeds. To prevent that a random number is repeatedly generated until the pot chosen is not empty. Since the game ends when either computer or player has all pots empty, this should not cause an infinite loop.

Evidence of testing and debugging in the development of the game.



Game in mid-play with player having chosen pot 1 which is emptying and has 3 left. The score is 6 – 0 to the computer.

When the teacher played he managed to win quite often but my cousin generally lost. The teacher strategy was to avoid a capture by the computer. My cousin tried each time to get a score.
Both of them enjoyed playing the game. The teacher noticed something that neither I nor my cousin had noticed. On a very few occasions the computer did not choose the pot that gives the most score. I finally tracked this down to an error in counting round the board but it was too late to put it right.

This criterion was awarded 12 marks. The game demonstrated evidence of original thinking, the use of complex algorithms and citation of sources through a link to the appendices.