purple mash

Computing

Scheme of Work

Unit 6.5 Text Adventures

Year Group: 6
Number of Lessons: 4

From 2simple

Contents

Introduction

This unit follows on from the Year 6 Coding unit (6.1). 2Code is used within this unit, and it is assumed that children have completed the coding unit to understand the coding concepts referred to in this unit. The last lesson of the Year 6 Coding unit (6.1) introduces text-based adventures and children have the chance to edit an existing text adventure. Lesson 3 of this unit uses the same text adventure; if children have recently completed this unit, they will be able to spend less time on Lesson 3 and have more time to design their own adventures based upon it.

These lesson plans make use of the facility within Purple Mash to set activities for pupils which they can then complete and hand in online (2Dos). This enables you to assess their work easily as well as distribute resources to all pupils. If children have not opened 2Dos before, then they will need more detailed instructions about how to do this. If your pupils do not have individual logins for Purple Mash, we can help you with this. Contact your school Purple Mash administrator or email us at support@2simple.com.

A teacher's guide to 2Dos can be found in the Teacher section: 2Dos Guide.

To force links within this document to open in a new tab, right-click on the link and then select 'Open link in new tab'.

Medium Term Plan

Lesson	Aims	Success Criteria
1	To find out what a text adventure is. To plan a story adventure.	• Children can describe what a text adventure is. • Children can map out a story-based text adventure. • Children can use 2Connect to record their ideas.
2	To make a story-based adventure.	• Children can use the full functionality of 2Create a Story Adventure mode to create, test and debug using their plan. • Children can split their adventure-game design into appropriate sections to facilitate creating it.
3	To introduce map-based text adventures.	• Children can map out an existing text adventure. • Children can contrast a map-based game with a sequential story-based game.
4	To code a map-based text adventure.	• Children can create their own text-based adventure based upon a map. • Children can use coding concepts of functions, two-way selection (if/else statements) and repetition in conjunction with one another to code their game. • Children make logical attempts to debug their code when it does not work correctly.

Lesson 1 – What Is a Text Adventure? Planning a Story Adventure

Aim

- To find out what a text-based adventure game is and to explore an example made in 2Create a Story.
- To use 2Connect to plan a 'Choose your own Adventure'-type story.

Success criteria

- Children can describe what a text adventure is.
- Children can map out a story-based text adventure.
- Children can use 2Connect to record their ideas.

Resources

Unless otherwise stated, all resources can be found on the main unit 6.5 page. From here, click on the icon to set a resource as a 2do for your class. Use the links below to preview the resources; right-click on the link and 'open in new tab' so you don't lose this page.

- Examples of the 'Choose your own Adventure' books would be useful but not essential. If children have any of these at home, they could bring them in to use for ideas and prompts for their own adventures.
- Red Riding Hood Adventure Game. For use on the whiteboard.
- Story Plan for Red Riding Hood.

Activities

1. What is a text adventure? Children will hopefully remember that a text adventure is a computer game that uses text instead of graphics.
2. They were very popular before graphics-based games were invented. Despite the lack of graphics, they are still fun to play and often require you to think and solve puzzles. Modern games such as 'escape the room' adventures are much like text adventures, but with the addition of visual clues.
3. Discuss the 'Choose your own Adventure' books and talk about children's experiences of these. Have they played computer games that follow a story? Most computer games have a story, and many take you on an adventure determined by the choices that you make. The game is designed in a way that reacts to the choices that you make and gives you a selection of options to choose from.
4. Many adventure games today are based upon books or films.
5. The theme can be quite simple. For example, a scenario of walking to the shops to run an errand and being presented with choices such as do you continue to the shops or play in the park? The main idea in a story adventure is a narrative where you are presented with choices that have different consequences.
6. Before making a story adventure, you need to plan it. In Purple Mash, you can use a tool called 2Connect to do this. See whether pupils remember using it before.

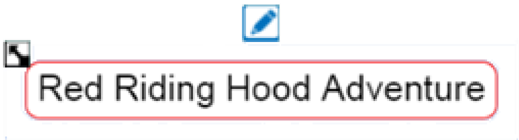
7. Show the Red Riding Hood example and talk through a few options for the story adventure. For example, what happens if she chooses to play in the woods? Which path does the traditional story take? Draw children's attention to the colours: choices are coloured red, and the story endings are purple. The yellow node shows a decision that will take you to a different storyline. Children should use a similar system to this as it will make coding their adventure much more straightforward.

8. Now open a blank 2Connect file and show children how to use it if they have not used it before.

9. In 2Connect, you can click and type anywhere in the document. If you click on the page and a node appears that you don't want, it will disappear if you don't write in it. The boxes of writing are called **nodes.**

10. Click somewhere near the top of the document and write the title.

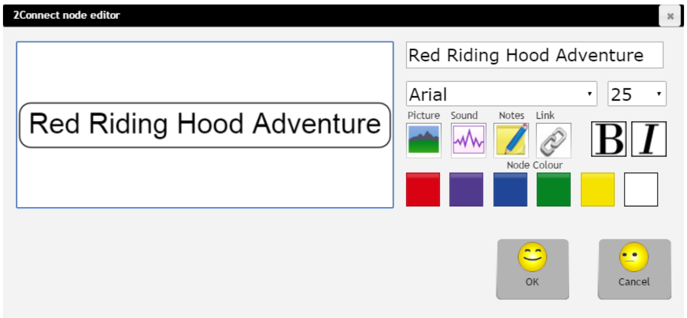
11. When you click on a node a pencil will appear, as will arrows in the top corner.



Red Riding Hood Adventure

12. The arrows allow you to make the node bigger and smaller.

13. To add a background colour to a node, click on the pencil and then choose the colour.



2Connect node editor

Red Riding Hood Adventure

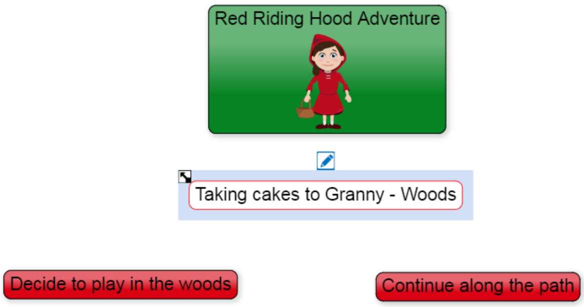
Red Riding Hood Adventure

Arial 25

14. Alter the text size so that all the nodes can fit on the screen.

15. You can easily add pictures, weblinks and sound to each of the nodes from this screen, though this is not strictly necessary for a text-based adventure.

16. Once you have added a title, add the next steps in the story.



Red Riding Hood Adventure

Taking cakes to Granny - Woods

Decide to play in the woods

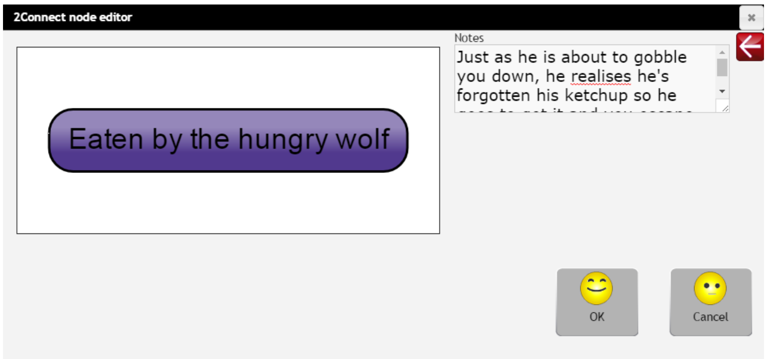
Continue along the path

17. To add a link, hover the mouse over the edge of the selected node where the border is pale blue. The arrow will turn to a hand. Click and drag the line to the node you wish to link. To remove a link, click on the small purple box, drag it away from the box and drop it on the blank page somewhere.

18. To change the direction of the link or colour of the line, click on the arrow on the line.

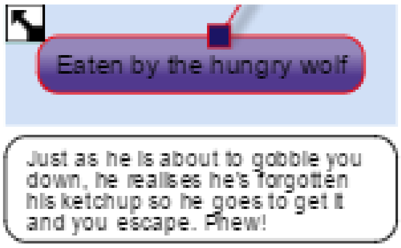
19. You will need to rearrange the boxes as you go to make the diagram look clear. Nodes can be dragged around and the links will remain attached.

20. Notes allow the user to add some information to the node; this can be useful as only a few characters of text can be inserted in a node. These appear when the node is clicked on. Click on the pencil and then on the ✎ button.



Notes

Just as he is about to gobble you down, he realises he's forgotten his ketchup so he



Just as he is about to gobble you down, he realises he's forgotten his ketchup so he goes to get it and you escape. Phew!

21. Before starting their plan, children should decide upon a theme for their story; perhaps a famous historical figure that you are studying would make a good main character? Their stories should contain just a few choices to begin with. They will be coding the stories next lesson and the more choices, the more complex the code. Suggest that they start with a simple story to get the idea of how to do it or decide in advance that they will only aim to create a portion of their story next lesson.

22. Once the plans are finished, they should be printed for the next lesson in which the children will be making their story using 2Create a Story and using the plans as a working document to structure the story.

Lesson 2 – Making a Story-based Adventure Game

Note for teachers:

If you have used this unit in the past you will notice that this lesson has been updated to use the new Adventure Mode in 2Create a Story instead of creating the text adventure using 2Code. If you wish to use 2Code, the old lesson plan is reproduced in Appendix 4. This can also be used to provide an extra challenge to stretch good coders.

Aim

- To use 2Connect to plans to create a story adventure using 2Create a Story.

Success criteria

- Children can use the full functionality of 2Create a Story Adventure mode to create, test and debug using their plan.
- Children can split their adventure-game design into appropriate sections to facilitate creating it.

Resources

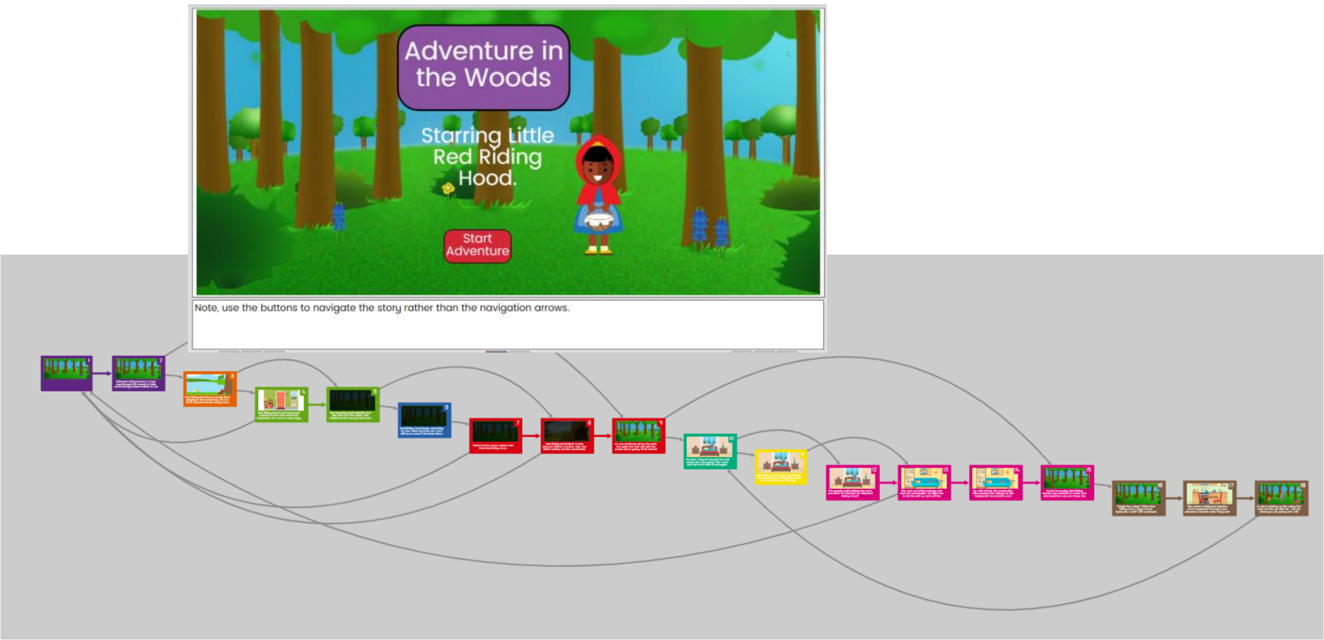
Unless otherwise stated, all resources can be found on the main unit 6.5 page. From here, click on the icon to set a resource as a 2do for your class. Use the links below to preview the resources; right-click on the link and 'open in new tab' so you don't lose this page.

- Children's completed and **printed** Red Riding Hood planning sheets from last lesson.
- Red Riding Hood Adventure Game . You might wish to set this as a 2do so that children can refer to it when making their stories.
- 2Create a Story Tool.

Activities

1. Explain that in this lesson, children will be using their plans to make the adventure games using 2Create a Story Adventure mode.
    2. Have children used 2Create a Story before to make e-books? Discuss what an e-book is. The functionality of 2Create a Story allows you to easily make a create-your-own-adventure style book. Even though the emphasis of this unit is on the text adventure aspect, you can also create animations and sounds to make the adventure more exciting.
2. Open the example story on the whiteboard and demonstrate the following to familiarise children with the tool. NB You might wish to play ▶ and try out the story first (try clicking on the characters as well).

3. Point out the drawing area and the text area.



Note, use the buttons to navigate the story rather than the navigation arrows.

4. Click on the overview button . This shows the story planner view which shows how the story flows. When debugging your story, it can be useful to check that each page has the expected number of links coming from it. You can also click on it to go straight to a page (even in play mode) and zoom in using a mouse wheel.

5. The play buttons; . This button appears both in the top and bottom menu-bars. The top bar button will 'play' the whole story from page 1. The bottom bar will open the current page in a preview window and play any associated sounds and animations.

6. Backgrounds; . Use this button to add backgrounds. You can upload images, use clip-art and edit uploaded images. Look at the pages of the example book. You will see that the same forest back ground has been used several times. On page 5, a filter ( ) has been used to make the same background look like night time. On page 10, the background has had wolf added to it. On page 3, Red Riding Hood’s cape has been drawn onto the background.

7. Sounds; some sounds have been added to the example book, they can be set for pages or for when a sprite is clicked on. Advise pupils to add these as extras at the end otherwise they will not finish the story. When making a story for a young child it is very effective to record the words of the story for each page. You can even compose your own music using the piano button in the sound picker tool.

8. Buttons; These are used throughout the story to navigate to the desired page. Look at the example book and double click on the buttons to see how the different options are used. The most important being ‘Go to Page’.

9. Sprites; . These are the characters in the story. Look at the sprites in the example story. They can be animated to achieve a variety of effects, they can also respond to mouse clicks. On page 10, the wolf's and Red's animations have delays set to make it look like he is getting into bed just as Red arrives. On several pages, the sprites clipart have been edited to change the facial expression on the characters.

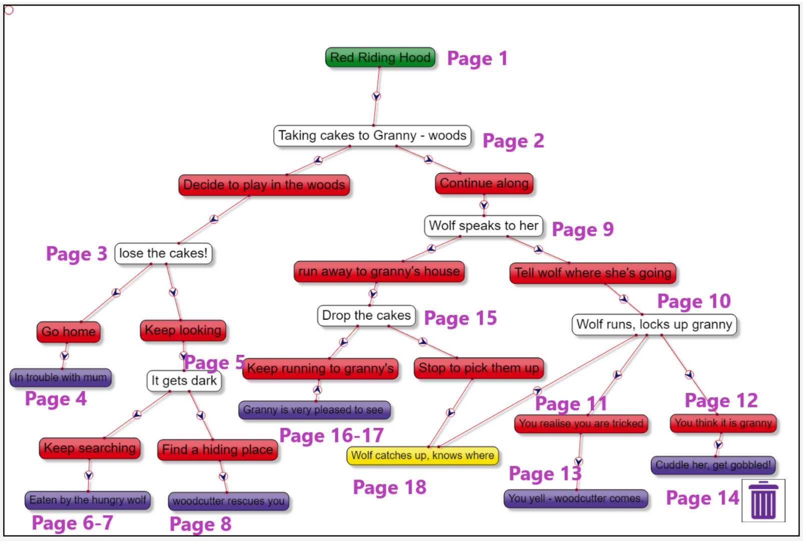
10. The bottom menu-bar contains various editing tools:

- The Arrow buttons navigate between the pages of the book. Pressing the right arrow on the last page will add a page.
- Text button: click here to edit the font, size and colour of the text. A screen will pop up allowing you to do this.
- Copy Page button: this will copy the current page. You can use this to make duplicate pages and then edit them individually so that the whole picture does not need to be redrawn each time. Many of the pages in the example were started this way.
- Paste Page: the clipboard will be coloured brown when there is something that can be pasted. Go to the page on which you want to paste the copied page and then click the button. A confirmation message will make sure that this is what you want to do to avoid accidentally overwriting pages. All animations and sounds will be copied.
- Delete Current Page button: a confirmation box will check that this is what you want to do.
- Clear Current Page button: a confirmation box will check that this is what you want to do. This will clear the content but not that page itself.

11. Once you have gone through these options. Children should open 2Create a Story and select 'My Adventure Story'.

12. Children should use the printed 2Connect plans to guide them. Each of the white nodes will be a page and the red nodes will be the buttons. The arrow to the next white node shows where the button should link to. The blue nodes are ending pages which will have a restart button on them to go back to page 1.

13. As they create the pages, children will find it helpful to add the page numbers. Here is such a plan for the example program. You can see a bigger version of this in Appendix 5



14. Children should save regularly and test and debug the story as they go. Encourage children to make use of the copy and paste page function to save time making each page from scratch.

15. Once the games are created, share them on a displayboard or blog so that others can enjoy them.

Lesson 3 – Introducing Map-Based Text Adventures

Aim

- To introduce an alternative model for a text adventure which has a less sequential narrative.

Success criteria

- Children can map out an existing text adventure.
- Children can contrast a map-based game with a sequential story-based game.

Resources

Unless otherwise stated, all resources can be found on the main unit 6.5 page. From here, click on the icon to set a resource as a 2do for your class. Use the links below to preview the resources; right-click on the link and 'open in new tab' so you don't lose this page.

- Example Y6 Text Adventure. Set this as a 2do for your class to open.
- Text Adventure Planner; this should be set as a 2Do for pupils to record their game designs.
- Pencils and paper for children to sketch out plans and ideas.

Activities

1. Discuss the adventure game that they explored during Unit 6.1. Let the children spend some time playing the 2Code adventure game if they have not played it for a while. Encourage them to map out the rooms like they did when they were working on the unit.

2. How does this adventure differ from the story-based adventures?

   - It is not sequential; you can go backwards as well as forwards.
   - You must solve a puzzle rather than take a route.
   - You either solve the puzzle and finish the game or you don't – there is no alternative ending.

   Map-based adventures don't ***have*** to do any of these things; it is up to the designer how they develop the game.

3. Next, ask the children to consider the code. If it has been a while since you completed unit 6.1, use the notes from Lesson 6 to guide the children through the code. This lesson is included at the end of these plans in Appendix 2.

4. Now is the chance for them to create their own map and story. The rest of this lesson should be spent planning their scenario, planning the quest and drawing a map for the game. For most children, a very simple game is a good starting place. The example game has six rooms, but you could have fewer if you wanted. For some children, aiming to navigate through four rooms without collecting any items is enough. Some children will want to make a complex game straight away, but as the number of rooms increases, the game gets much harder to code as there is so much more to keep track of.

5. The Text Adventure Planner document can be used to note ideas and upload images. If children draw a map for their game freehand, they can take a photo of it and upload this to their computer by whichever method you use in your school (device camera, separate digital camera). Then, they could insert it into the front cover of the document. To add pictures from their computer to the picture boxes in the planner, click on the green cross buttons ⊕. Next, click the Add to Gallery button and

locate the file. There are controls to rotate and zoom in on parts of an image on this screen. When you are happy with the image, click Choose.

6. Children could also fill in the sections about the scenario and aim of the game.

Lesson 4 – Coding a Map-based Text Adventure

Aim

- To use written plans to code a map-based adventure in 2Code.

Success criteria

- Children can create their own text-based adventure based upon a map.
- Children can use coding concepts of functions, if/else statements and repeats in conjunction with one another to code their game.
- Children make logical attempts to debug their code when it does not work correctly.

Resources

Unless otherwise stated, all resources can be found on the main unit 6.5 page. From here, click on the icon to set a resource as a 2do for your class. Use the links below to preview the resources; right-click on the link and 'open in new tab' so you don't lose this page.

- Example Y6 Text Adventure. This is the same as last lesson.
- Text Adventure Planner; children should open the documents that they started last lesson.

Activities

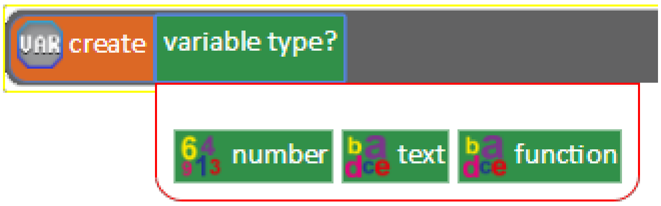
1. Children should refresh their memories on the game that they are planning to code. The aim for today is to get it working.
2. The first step is to decide upon what variables need to be tracked. In the example game, these are detailed in the table below.
3. Children can use and adapt the example game. Children will be making changes to their code, so it is worth talking about the need to save their files frequently after testing that they work. It can be easy to remove some vital code and then not know how to get the code back to how it was before, so by saving correct code after each successful change you can always go back to how it was by reopening your file. Also, highlight the Undo button, which appears when you have changed something so that you can undo it; often, this will fix the problem.

Variable name	Type	Purpose
haveDiamond	number	0 if the player doesn't have the diamond 1 if the player does
haveKey	number	0 if the player doesn't have the key 1 if the player does
room	number	The number of the room that the player is in
finished	number	0 if the player has not escaped 1 if the player has escaped
answer	text	This gets set to whatever the player types in

4. Children should detail the variables in the planning document; they might realise that they need to change things as they go along and having a written record will help them to work out what they need. Encourage them to make changes to debug their game.
5. The next part of the code involves the functions for each room. You might need to remind children what **functions** are and how they work in 2Code:

A function is a section of code that gets run when it is called from the main code. A function in a program is usually a piece of code that gets run lots of times. **In the adventure games, there is a function for each room that gets called when that room is entered.**

Functions are usually put at the beginning of the code.
To create a function in 2Code, drag in the 'create variable' block and then select the 'function' option.

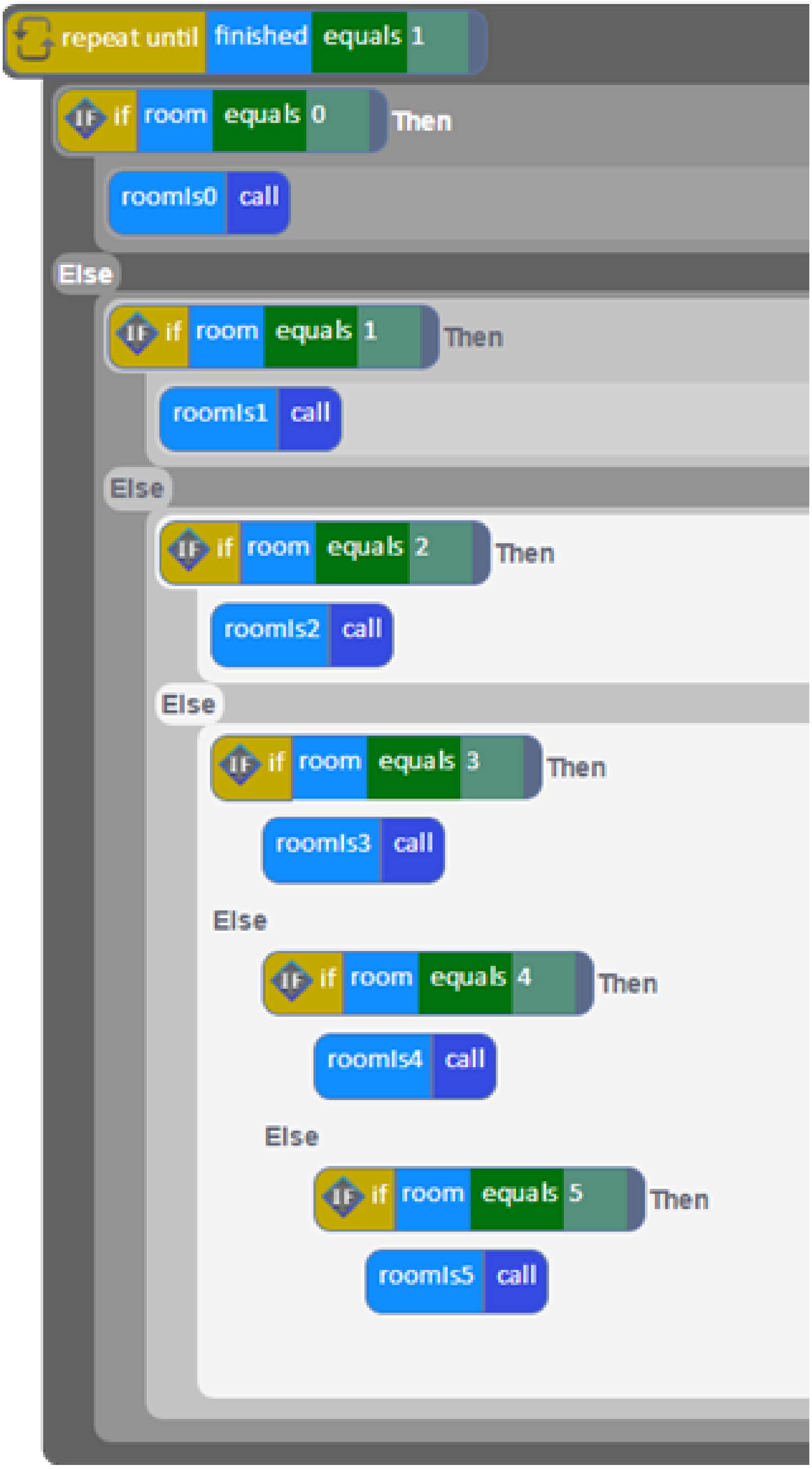


It is very important to give your function a sensible name which lets you know what it does, otherwise your code will get very confusing.

6. Encourage children to use and adapt the existing functions in the example program. They will need to use their maps to decide which prompts need to appear in each function and what the room variable should be set to, depending upon the direction in which the player enters.
7. There is space in the planning document for children to add some screenshots of their code. One way to create these is to use the Windows Snipping Tool (type 'snipping tool' in the search bar on your computer to locate this). This enables you to capture a section of the screen and save it as an image file on your computer, then you can add the image using the green cross button in the planning document.

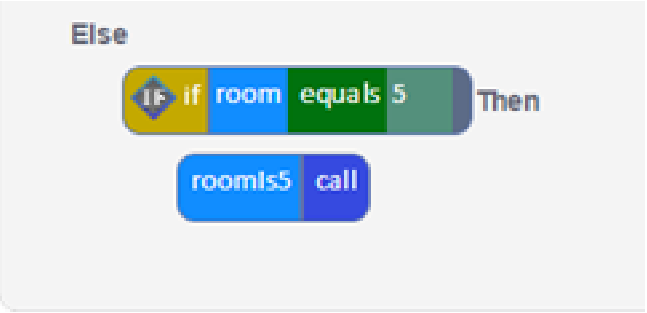
8. Next, it is time for children to adapt the main code that calls the functions.

   In the example, this looks like:



```
repeat until finished equals 1
    if room equals 0 Then
        roomIs0 call
    Else
        if room equals 1 Then
            roomIs1 call
        Else
            if room equals 2 Then
                roomIs2 call
            Else
                if room equals 3 Then
                    roomIs3 call
                Else
                    if room equals 4 Then
                        roomIs4 call
                    Else
                        if room equals 5 Then
                            roomIs5 call
```

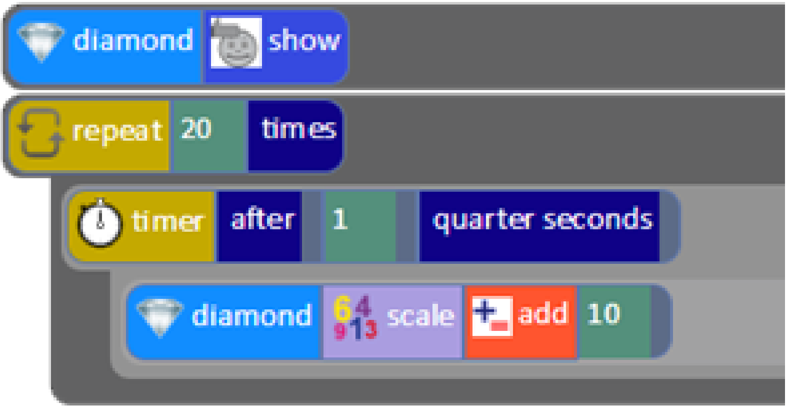
If the new adventure has six rooms and the functions have the same names as in the example, then this code does not need to be changed. If children have more rooms, they will need to add more if/else statements inside the last 'Else' statement:



```
Else
    if room equals 5 Then
        roomIs5 call
```

If they have fewer rooms, the simplest thing is for the code to remain as it is and some of the code will not be used.

9. The next stage is to adapt the conclusion part of the code:



```
diamond show
repeat 20 times
    timer after 1 quarter seconds
        diamond scale add 10
```

Once they have done this, they can complete the Conclusion section in their planning document.

10. Lastly, they will need to test and debug their code. This could take longer than all the other stages, depending upon what is going wrong. Children could open their own code in one tab and open Purple Mash in another tab with the example code to compare both of them to see what could be going wrong. Encourage children not to be disheartened if their code does not work and they get confused; they are acting like true coders by doing this. Most coders spend much of their time debugging and solving problems with the programs that they write. Encourage them to celebrate each little step towards finding a solution. Hopefully, they will feel their brains growing while they are doing this!

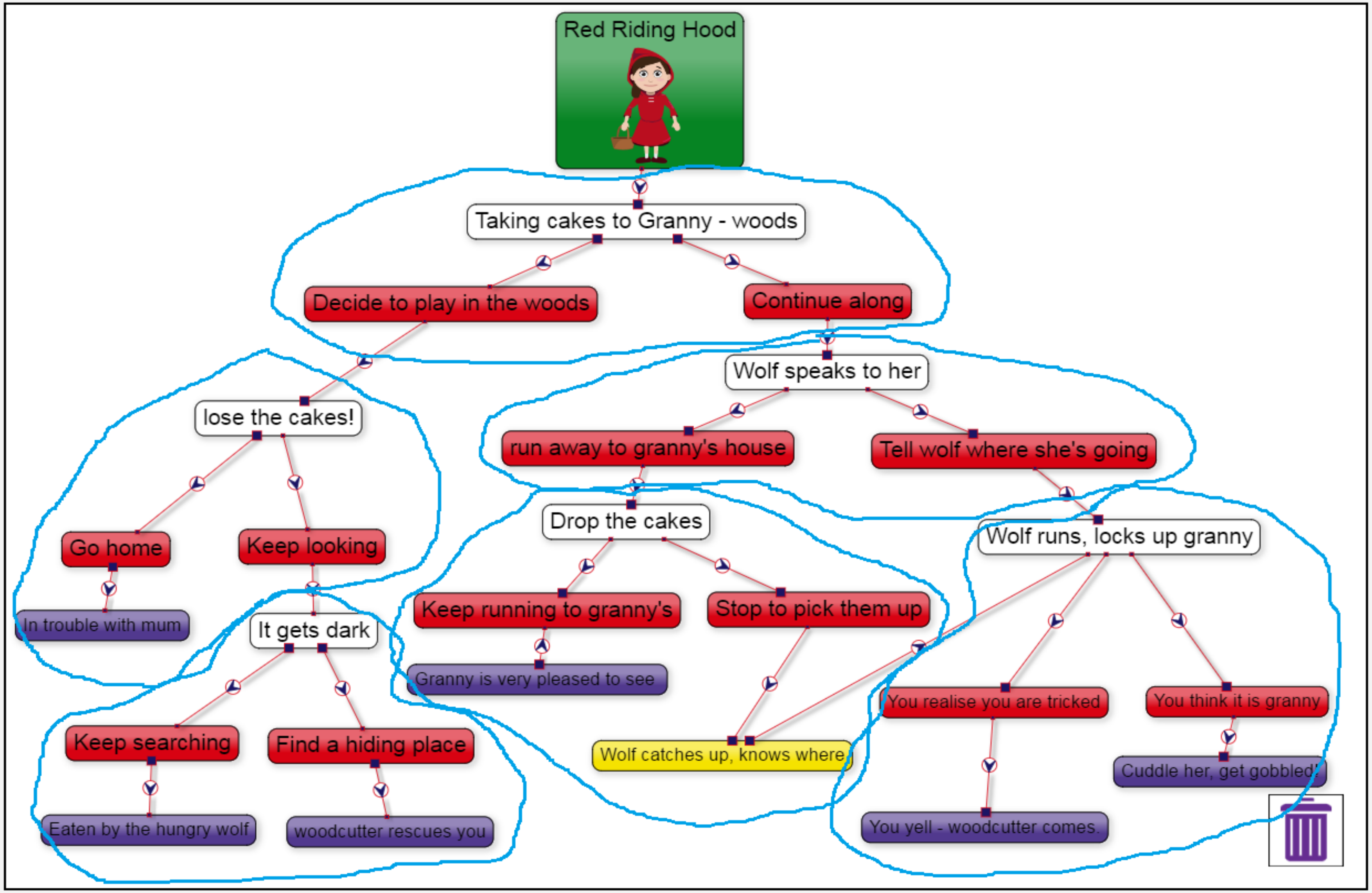
11. The last section of the planning document is for sharing their code. Children can add a QR code and link to their game for others to play. If the planning documents are printed and displayed, scanning the QR code with a tablet will take you straight to the game in Play mode. Clicking on a link in an electronic copy of the file will also take you to a playable version of the game.

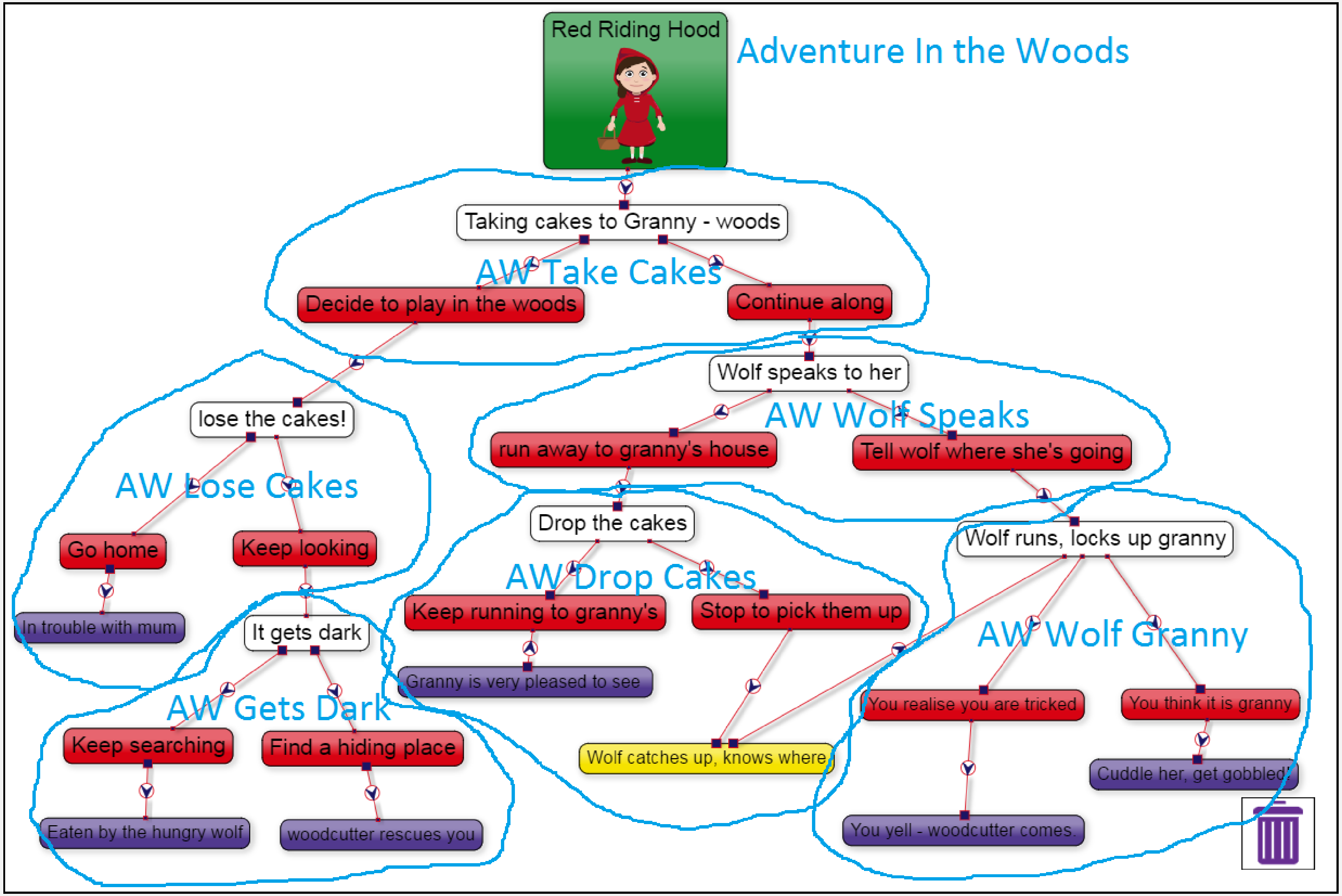
Try sharing the games using **2Blog** to make a class blog, which other children in the school can read and comment upon. Please share your finished games with us at 2Simple as well; we'd love to see them!

To **share** a game, first save it in an online folder. Then, click on the 🌐 button. You have several options for sharing here, including sending an email and blogging. Click on ‘**Share’**, then ‘**Link & QR code**’. Children can click ‘Copy’ to copy the link and then paste it into their file by clicking in a text box and pressing the ‘Ctrl’ and ‘V’ keys on their computer.

Clicking on the **QR code** will open a bigger version; right-click on this and save it to your computer, then import it into your planning file.

Appendix 1: 2Connect Diagrams




Red Riding Hood
Adventure In the Woods
Taking cakes to Granny - woods
AW Take Cakes
Decide to play in the woods
Continue along
lose the cakes!
AW Lose Cakes
Go home
Keep looking
In trouble with mum
It gets dark
AW Gets Dark
Keep searching
Find a hiding place
Eaten by the hungry wolf
woodcutter rescues you
Wolf speaks to her
AW Wolf Speaks
run away to granny's house
Tell wolf where she's going
Drop the cakes
AW Drop Cakes
Keep running to granny's
Stop to pick them up
Granny is very pleased to see
Wolf catches up, knows where
Wolf runs, locks up granny
AW Wolf Granny
You realise you are tricked
You think it is granny
Cuddle her, get gobbled!
You yell - woodcutter comes.

Appendix 2: Unit 6.1 Lesson 6 – Using 2Code to Make a Text-based Adventure

Aim

- To explore how 2Code can be used to make a text-based adventure game.

Success criteria

- Children can follow through the code of how a text adventure can be programmed in 2Code.
- Children can adapt an existing text adventure to make it unique to their requirements.

Resources

- Example text-based adventure game from the Year 6 examples. Set this as a 2do for your class to open.
- Pencils and paper for children to sketch out plans and ideas.

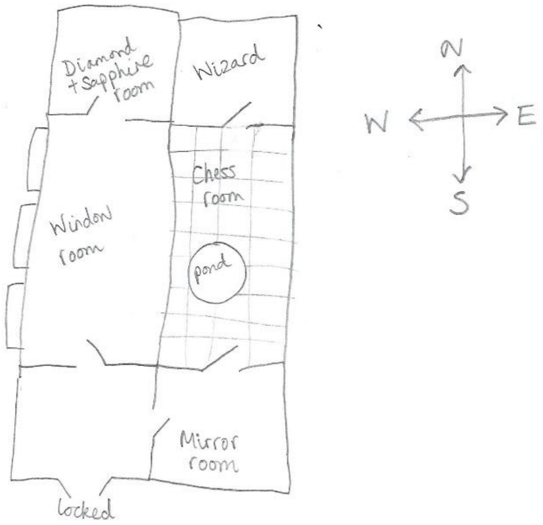
Activities

1. What is a text adventure?
2. Explain that a text adventure is a computer game that uses text instead of graphics. The games simulate environments in which players use text commands to control characters and influence the environment. A bit like computer versions of the 'Choose your own Adventure' books, but often with puzzles to solve.
3. They were very popular before graphics-based games were invented. Despite the lack of graphics, they are still fun to play and usually require you to think and solve puzzles. Modern games such as 'Escape the room' adventures are much like text adventures, but with the addition of visual clues. Text adventures are also known as 'Interactive fiction', where they are more of a story with choices than puzzles to solve.
4. We are going to begin today by playing a simple text adventure written using 2Code. Then, you will have a chance to change the code to make the adventure your own.
5. To solve this adventure, it is very useful to sketch a map of the places that you go and write some notes. It will also help you to adapt this adventure yourself.
6. Show children how to open the adventure game and press Play to get started. For the sake of simplicity of programming, answers can only be typed in lower case. Directions should be entered as: 'n', 'e', 's' or 'w'.
7. Give the children some time to play the adventure; see who is first to solve it.

> The adventure has only six rooms, but let children explore freely without clues. Children should go to the room with the diamond on the floor and take it to the wizard. He gives you the key which you can use to exit the game.

8. Once children have had time to play, compare the maps that they have drawn and discuss how having a map is helpful for solving the game.

9. It is useful for everyone to have a simple map when discussing the code. Something like this is perfect:



10. The next stage is to look at the code. Children will be making changes to their code, so it is worth talking about the need to save their files frequently after testing that they work. It can be easy to remove some vital code and then not know how to get the code back to how it was before, so by saving correct code after each successful change you can always go back to how it was by reopening your file. Also, highlight the Undo button, which appears when you have changed something so that you can undo it; often, this will fix the problem.

11. In the code, each room has been given a number. Tell the children that you will look through the code together and try to write the numbers on the map.

12. Look at the first five lines of the code; this is where various variables are set. Do the names of the variables provide clues as to what they are for? They should do, and it's the reason that you should always try to name variables well. Once your coding gets more complex, it is very confusing if the variables are all called things like 'MyNumber1' or 'MyNumber2'.

    See the following table of variables:

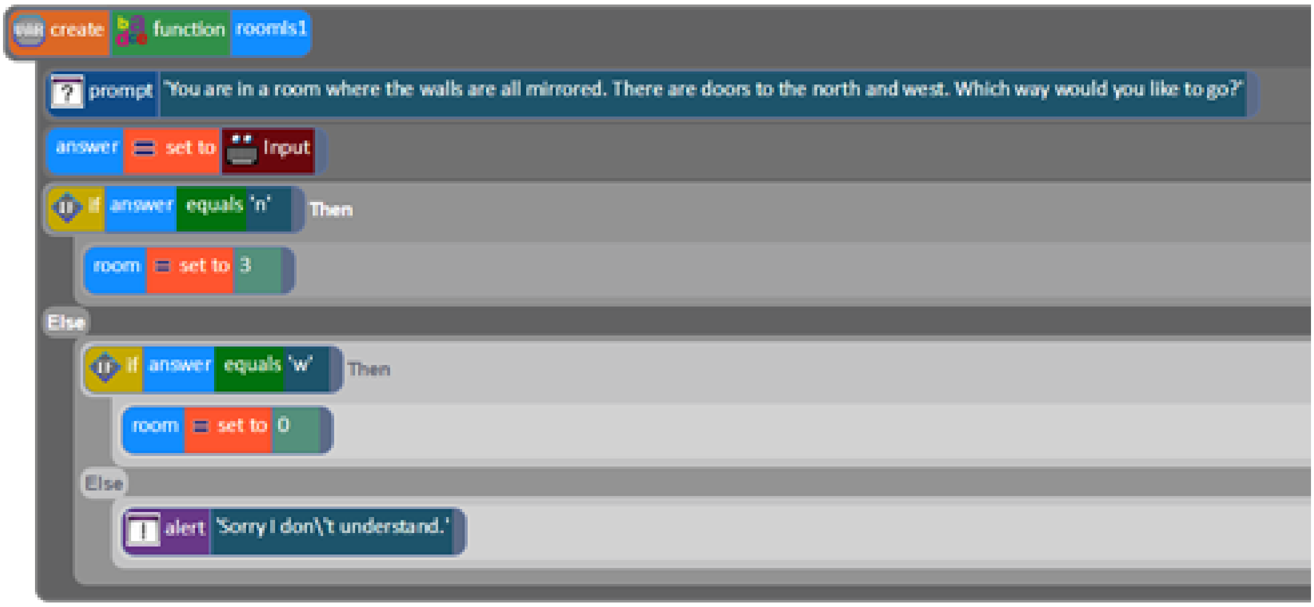
    The variables are:

Variable name	Type	Purpose
haveDiamond	number	0 if the player doesn't have the diamond 1 if the player does
haveKey	number	0 if the player doesn't have the key 1 if the player does
room	number	The number of the room that the player is in
finished	number	0 if the player has not escaped 1 if the player has escaped
answer	text	This gets set to whatever the player types in

13. **Functions**: The next lot of code involves functions for each room, called roomIs0, roomIs1 etc. The functions are only run when they get called by the main code. Inside each function is the code that should be run when the player is in this room. This includes the text about where the doors are, along with code to test if the player has collected the diamond or has the key, etc.

A picture of each of these functions can be found on the Unit 6.5 main page for your use, if you wish to use them. Some children might find it helpful to have printed versions that they can refer to when changing code.

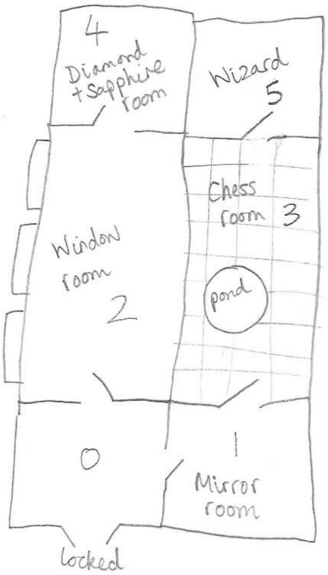
Here is the example for roomIs1



```
create function roomIs1
    prompt 'You are in a room where the walls are all mirrored. There are doors to the north and west. Which way would you like to go?'
    answer set to input
    if answer equals 'n' Then
        room set to 3
    Else
        if answer equals 'w' Then
            room set to 0
        Else
            alert 'Sorry I don\'t understand.'
```

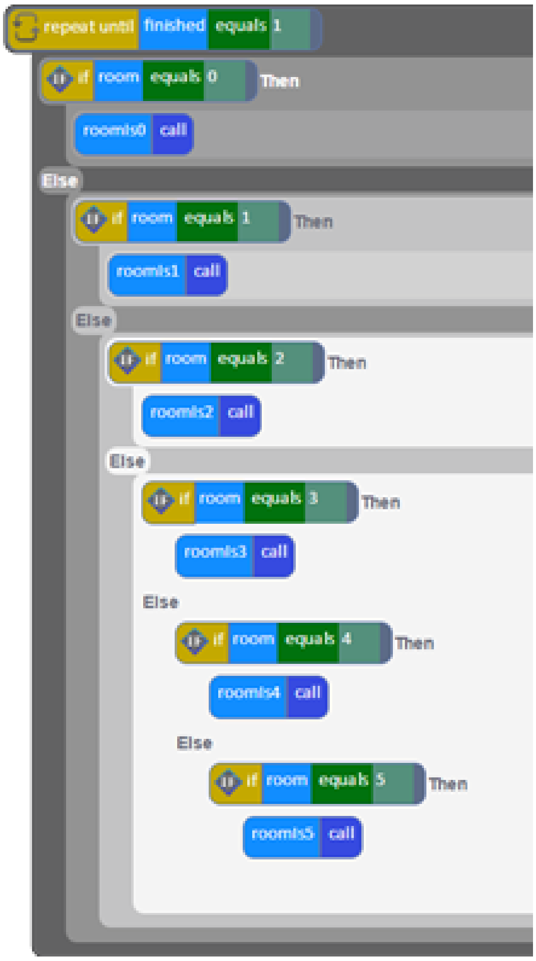
Can children use this information to label the rooms on their map with the room numbers? In the example above, the room number is 1 and you can tell from the text that this is the room with mirrored walls.

Children's maps should end up like this:



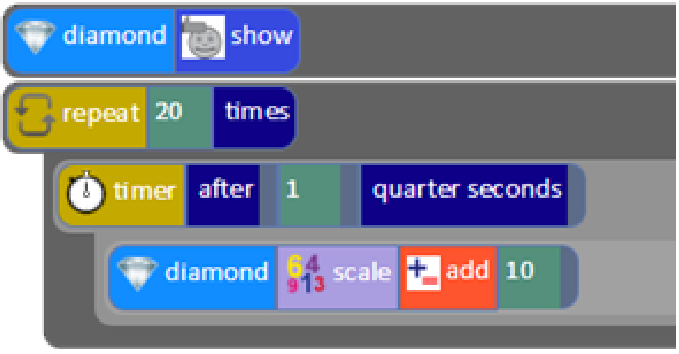
14. **Repeat**: The main code that runs when the program starts, comes after the functions. The first line of this says the following code should be repeated until **finished = 1**. Looking at the variables above, this means that it continues until the player has escaped.

    This code uses if/else statements to establish which room the player is in, and then runs the appropriate function for that room.



```
repeat until finished equals 1
  if room equals 0 Then
    roomIs0 call
  Else
    if room equals 1 Then
      roomIs1 call
    Else
      if room equals 2 Then
        roomIs2 call
      Else
        if room equals 3 Then
          roomIs3 call
        Else
          if room equals 4 Then
            roomIs4 call
          Else
            if room equals 5 Then
              roomIs5 call
```

Conclusion: This is the code that runs when finished = 1. This is when the payer has escaped and finished the adventure. Can children explain what it does?



```
diamond show
repeat 20 times
  timer after 1 quarter seconds
    diamond scale add 10
```

15. Ask children to change the room descriptions to their own ideas and then play the game again. How much can they make the game feel different just by changing the text?

NOTE: Sometimes when text is changed, there is a bug which reverts the text to what it was before when they go to change another part of the code. If this happens, children should drag another prompt for input **command below the existing one, enter the new text and then delete the original text by dragging it to the trash bin in the bottom right of the screen.**

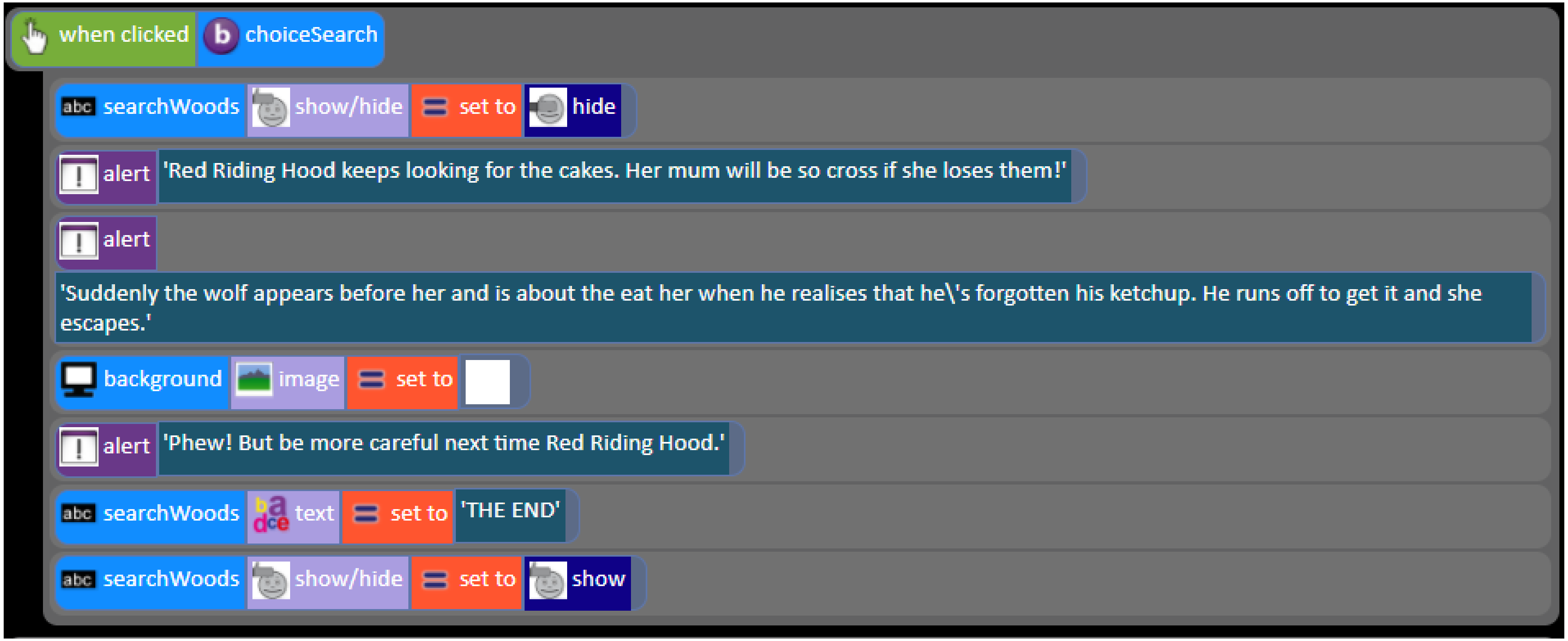
The diamond, wizard and key could become something else. Perhaps the player is Alice, lost in Wonderland, and she needs to give the wizard something so he can give her a shrinking potion? Perhaps the player is a master spy who needs to find a vital clue and report it back to the wizard\Q\prime minister?

16. Play the resulting games as a class and enjoy!

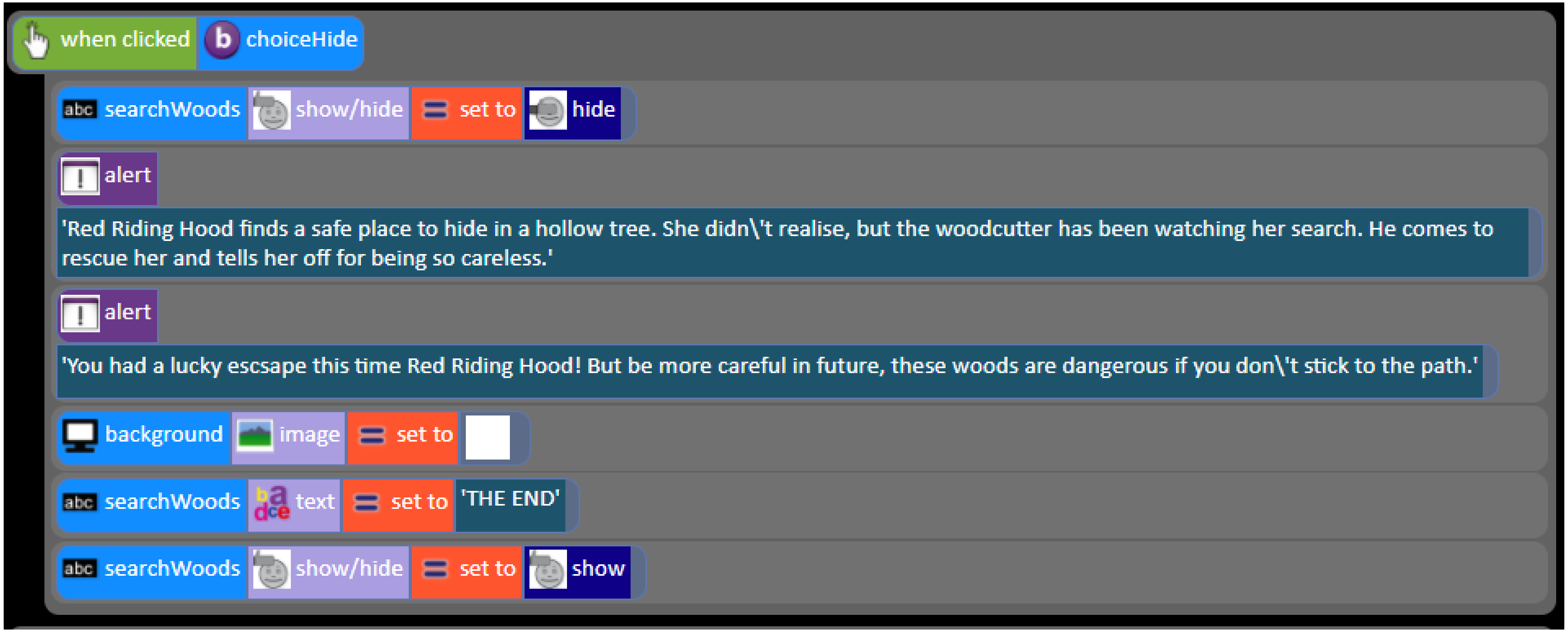
Extension

In Unit 6.5, children will have the opportunity to change the rooms in this adventure. Some children might be eager to try this out now. Encourage them to map out their rooms and start with small changes before building on them. Save working code often so that, if it all goes wrong, they don't lose everything.

Appendix 3 – Code snippets



```
when clicked  choiceSearch
    searchWoods  show/hide  set to  hide
    alert  'Red Riding Hood keeps looking for the cakes. Her mum will be so cross if she loses them!'
    alert
    'Suddenly the wolf appears before her and is about the eat her when he realises that he\'s forgotten his ketchup. He runs off to get it and she escapes.'
    background  image  set to
    alert  'Phew! But be more careful next time Red Riding Hood.'
    searchWoods  text  set to  'THE END'
    searchWoods  show/hide  set to  show
```

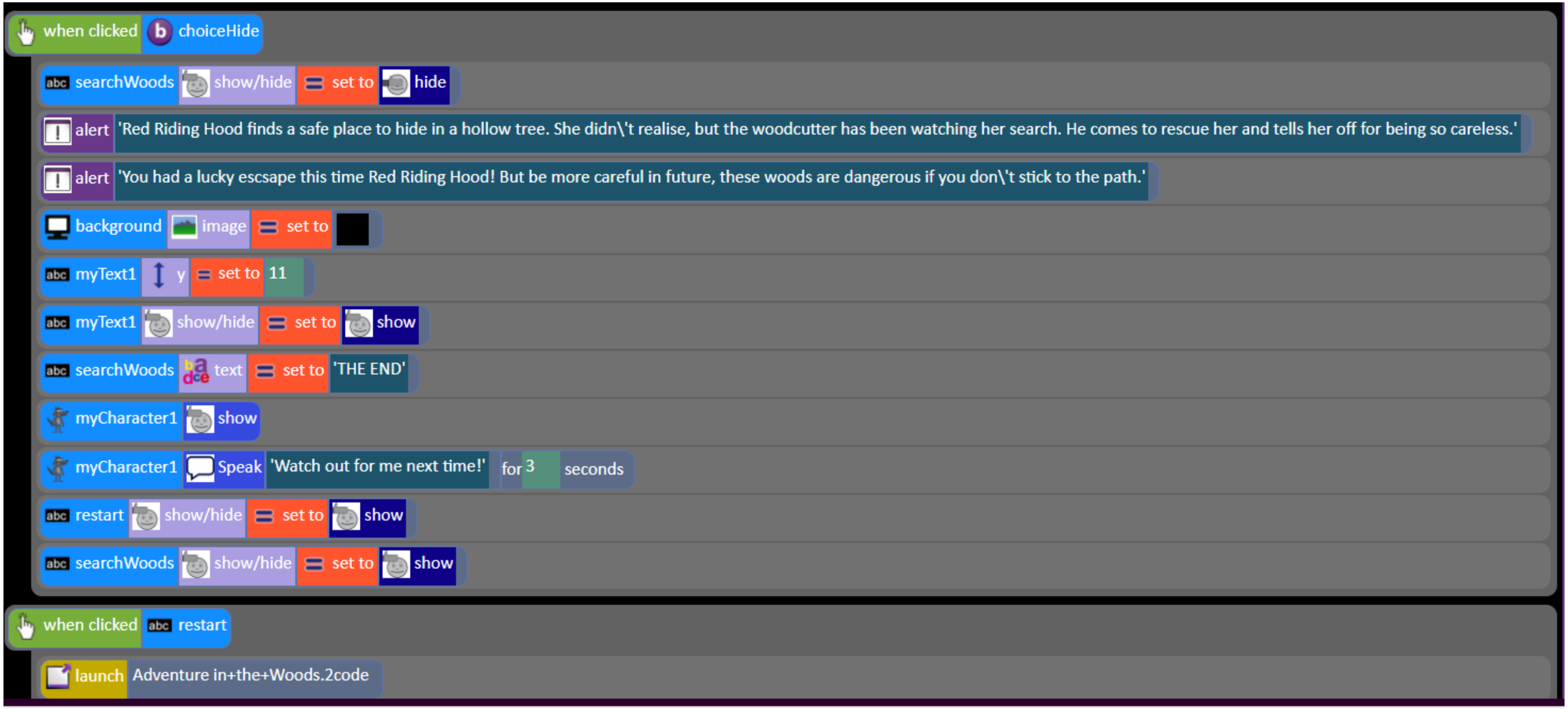


```
when clicked  choiceHide
    searchWoods  show/hide  set to  hide
    alert
    'Red Riding Hood finds a safe place to hide in a hollow tree. She didn\'t realise, but the woodcutter has been watching her search. He comes to rescue her and tells her off for being so careless.'
    alert
    'You had a lucky escsape this time Red Riding Hood! But be more careful in future, these woods are dangerous if you don\'t stick to the path.'
    background  image  set to
    searchWoods  text  set to  'THE END'
    searchWoods  show/hide  set to  show
```

How to make the buttons appear to hide

In this example, there is a text object called ‘myText1’. In design view myText1 is set to hide and has a black background. The text is lots of spaces to make the box big enough to cover the buttons.

The text object ‘restart’ functions as a button to restart the game by launching the first program.



```
when clicked  b choiceHide
    abc searchWoods  show/hide  = set to  hide
    alert 'Red Riding Hood finds a safe place to hide in a hollow tree. She didn\'t realise, but the woodcutter has been watching her search. He comes to rescue her and tells her off for being so careless.'
    alert 'You had a lucky escsape this time Red Riding Hood! But be more careful in future, these woods are dangerous if you don\'t stick to the path.'
    background  image  = set to ■
    abc myText1  y  = set to 11
    abc myText1  show/hide  = set to  show
    abc searchWoods  text  = set to 'THE END'
    myCharacter1  show
    myCharacter1  Speak 'Watch out for me next time!'  for 3  seconds
    abc restart  show/hide  = set to  show
    abc searchWoods  show/hide  = set to  show

when clicked  abc restart
    launch  Adventure in+the+Woods.2code
```

Appendix 4 – Old Lesson 2 – Making a Story-based Adventure Game using 2Code

Aim

- To use 2Connect to plans to code their own story adventure using 2Code.

Success criteria

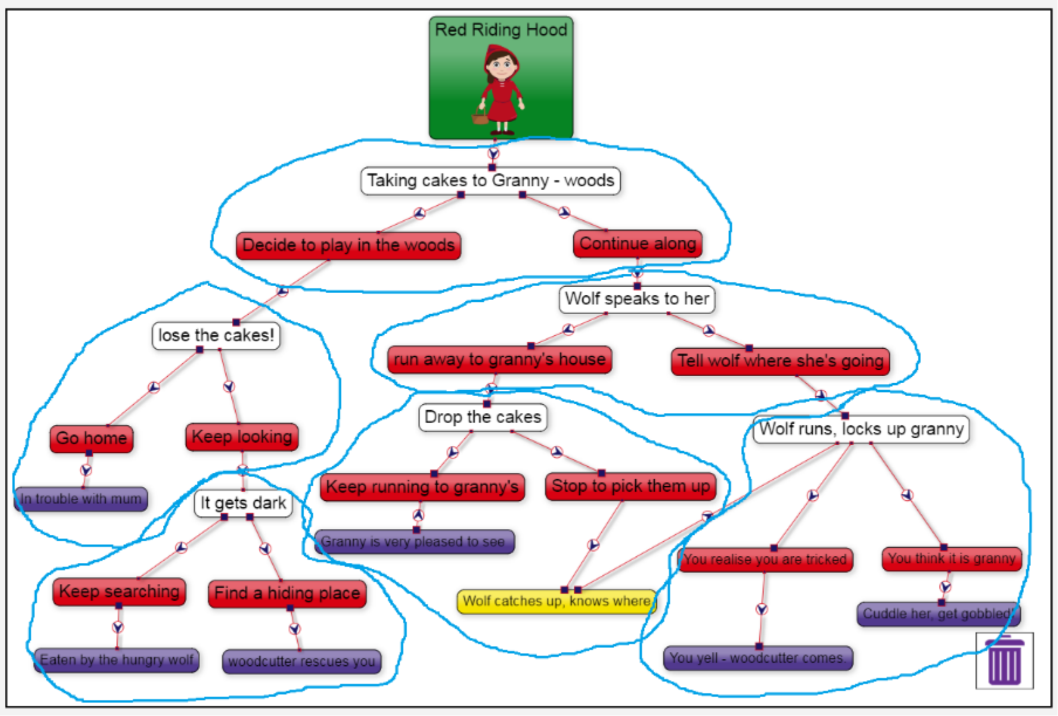
- Children can split their adventure-game design into appropriate sections to facilitate coding it.
- Children can code, test and debug the sections using 2Code.
- Children can use the 'launch' command in 2Code to bring all the sections of their game together into a playable adventure game.

Resources

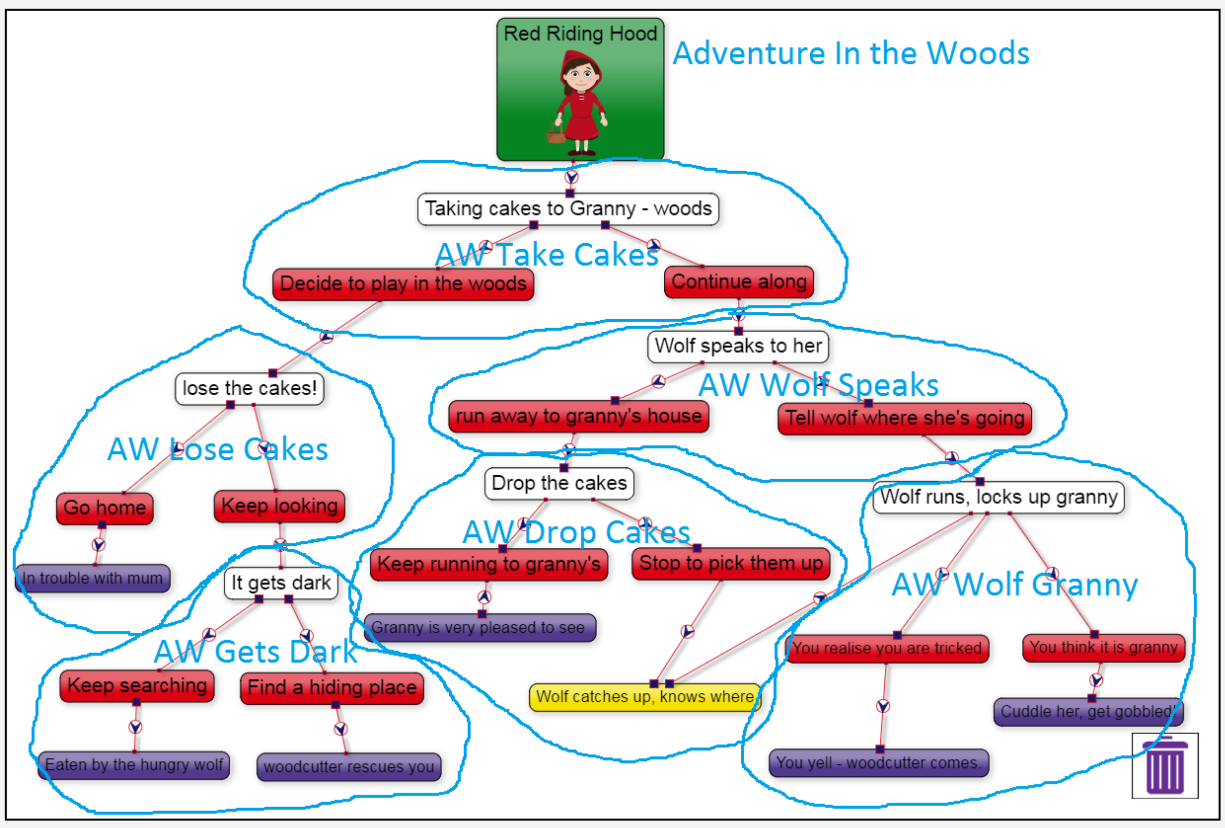
- Children's completed and **printed** Red Riding Hood planning sheets from last lesson.
- Diagrams of how to split up the 2Connect chart are repeated as large pictures in Appendix 1, so you can display them on your whiteboard if you wish.

Activities

1. Explain that in this lesson, children will be using their plans to make the adventure games using 2Code.
2. They will be using the 'launch' command within 2Code to launch a new program each time the player clicks on a choice. Can they remember using this command before? (They used it when making a showcase of their work in Unit 6.1.)
3. First, get the printed 2Connect plans and instruct children to split their plans into the separate choices made in the adventure. To do this, look at each of the white nodes and the choices (red) coming from them and draw a boundary around them, including any story endings. The example looks like this (a big diagram is in Appendix 1 for use on the whiteboard):



4. Each of these sections will be a separate 2Code program. To make launching the correct programs distinct from one another, give each of the programs a sensible name. Label each section with a suitable name as well, including the title screen. Here is the example:



5. Give children time to label their plans. Then, open 2Code Free Code Gorilla on the whiteboard and go into Design View.

6. It is easier to start coding the endings of the story and work backwards; this is because each earlier piece of code will launch a later one. In the example diagram above, we are first coding the program called 'AW Gets Dark'.

7. First, show children how to add a background for their story. This isn't strictly necessary in a text adventure but it adds some interest. However, children should not be spending time making their screens look pretty with animated graphics (except maybe for the endings), so discourage them from overcomplicating the design.

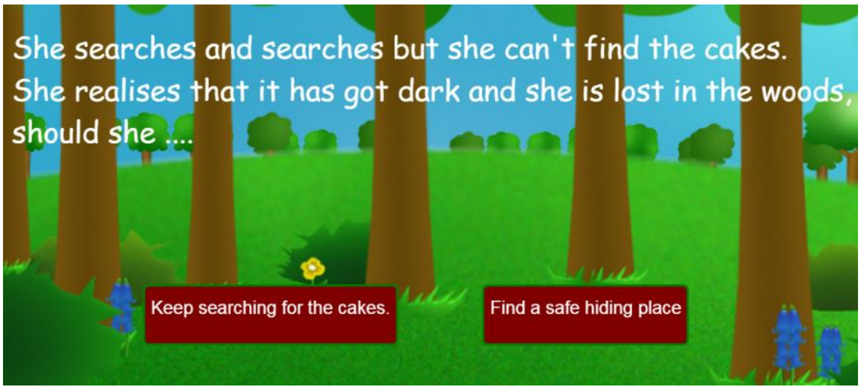
8. Click on the 🖥 background icon at the bottom of the left-hand panel.

9. Click on the ? in the image box image ? and choose something appropriate for the background. If you wish, children could design their own background using 2Paint or 2Paint a Picture, save it and import it into 2Code using the button.

10. Use the text objects abc text to add words to the screen (double-click on the object to edit the words) and edit the font, size, background colour and font colours on the side-bar on the left. To make coding less complicated, make sure that each piece of text has a descriptive name, e.g. 'lostCakes' or 'searchWoods'.

The text needs to be written in a way that leads to the player making a decision. For example, 'She searches and searches but she can't find the cakes. She realises that it has got dark and she is lost in the woods. Should she … or …?'

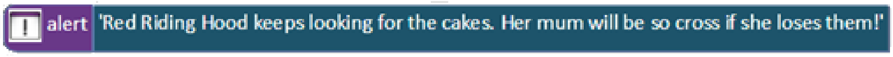
11. Drag a button object onto the screen and make it one of the choices buttons. Edit the writing on the button, colours and font size and then click on the Copy button. The copy will be on top of the original, so drag it to a new position. Remember to give the button objects sensible names, e.g. 'choiceSearch', 'choiceHide'.
12. The screen should end up with children's own versions of the example:



She searches and searches but she can't find the cakes.
She realises that it has got dark and she is lost in the woods, should she ....

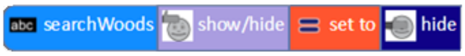
13. Give children a chance to create their screens, then come back together.
14. Exit Design View to begin coding what happens when the player clicks on one of the choices. In the example below:

- messages are sent to the player using the 'alert' commands



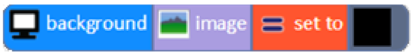
```
alert 'Red Riding Hood keeps looking for the cakes. Her mum will be so cross if she loses them!'
```

- the text is set to 'hide'



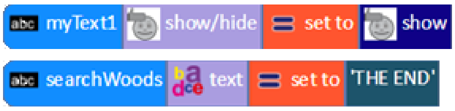
```
searchWoods show/hide set to hide
```

- and the background is set to a black screen.



```
background image set to ■
```

- The text message is then set to 'THE END' and the show/hide for this is set to 'show' once again.

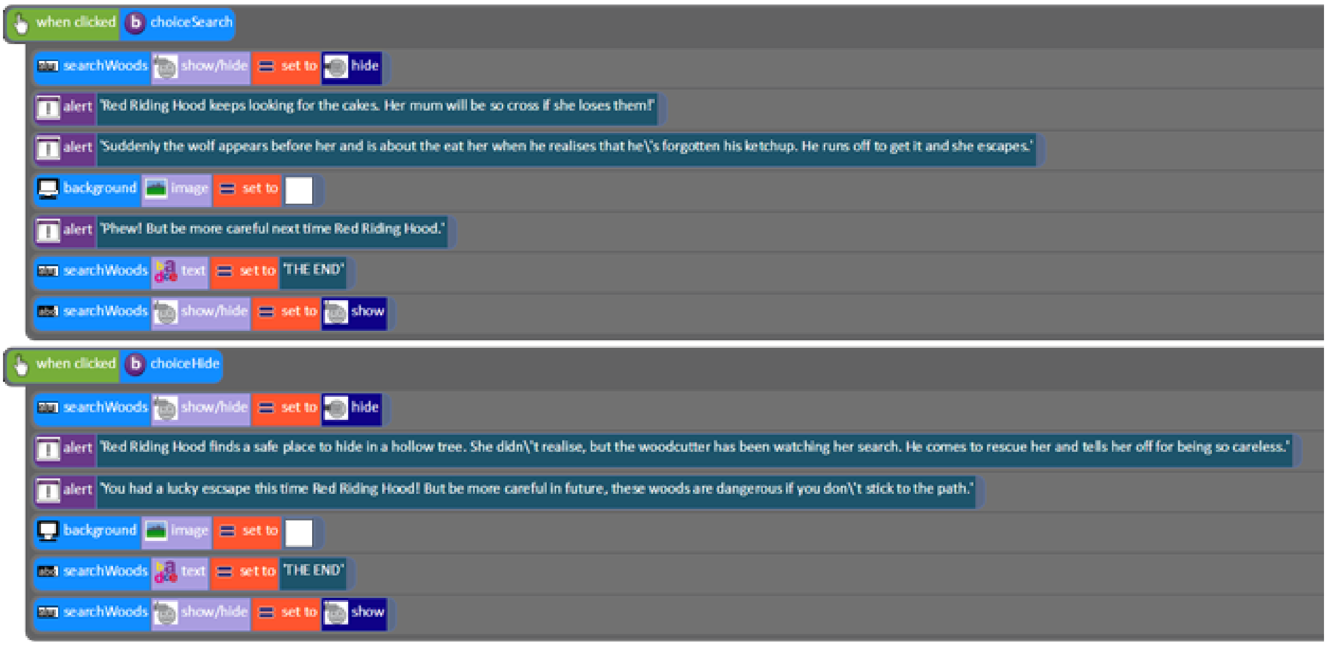


```
myText1 show/hide set to show
searchWoods text set to 'THE END'
```

Note: The black screen effect was achieved by selecting Paint Image in the background picker and then flood-filling the background in black.

NOTE: In the example story, the buttons appear to hide at the end of the story but in 2Code there is no 'Hide' command for buttons. The way to achieve this is to have an additional text object with a black background with the text set to lots of spaces to make the text box big enough to cover the buttons. Set this to 'hide' initially (in design view) and just show and move into position at the end when you want the buttons to hide. There is some example code that does this in Appendix 3. This is not a necessary step to make the program work but is more a design preference, you might wish to include this at the end once children have made a functioning program and are refining their work.

Here is the full example (without 'hiding' buttons) (larger examples are in Appendix 3):



```
when clicked choiceSearch
    searchWoods show/hide = set to hide
    alert 'Red Riding Hood keeps looking for the cakes. Her mum will be so cross if she loses them!'
    alert 'Suddenly the wolf appears before her and is about the eat her when he realises that he\'s forgotten his ketchup. He runs off to get it and she escapes.'
    background image = set to
    alert 'Phew! But be more careful next time Red Riding Hood.'
    searchWoods text = set to 'THE END'
    searchWoods show/hide = set to show
when clicked choiceHide
    searchWoods show/hide = set to hide
    alert 'Red Riding Hood finds a safe place to hide in a hollow tree. She didn\'t realise, but the woodcutter has been watching her search. He comes to rescue her and tells her off for being so careless.'
    alert 'You had a lucky escsape this time Red Riding Hood! But be more careful in future, these woods are dangerous if you don\'t stick to the path.'
    background image = set to
    searchWoods text = set to 'THE END'
    searchWoods show/hide = set to show
```

15. Children should be given time to code a similar example in their own adventure story and to test it to ensure that the right things happen when the buttons are clicked.

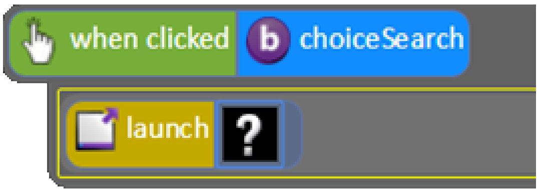
16. Make sure that the program is saved, and then use save as a second time to save it with a **new name.** This way, children can copy their existing program and then adapt it rather than starting from scratch each time.

    Note: for alert commands, it is better to drag in a new alert command and type the text than trying to edit the existing text as the update sometimes does not save and the text reverts to how it was before. Alerts that are not needed can then be dragged to the trash bin in the bottom right-hand corner of the screen.

17. Children should adapt their code to make the next program up in the chain. In the case of the example, it is called 'AW Lose Cakes'. In this code, one of the buttons needs to launch the 'AW Gets Dark' program.

18. This is how to do it:

19. First, drag the 'launch' command into the code when the button is clicked



```
when clicked choiceSearch
    launch ?
```

20. Next, click on ? then on Choose 2Code Activity and locate the saved program. You will get a little preview of the program, click on OK. Now the button will launch the program when it is clicked.

21. Children should code this program and test it before moving on to the other sections of the game.

22. **It is really important for children to check before saving that they are saving their files with the correct names, as it can be really easy to overwrite their programs. Show the children the option of saving to their computer as well as saving online, creating two copies of their files, in case they lose one.**

23. The Title screen (called 'Adventure in the Woods') will be the last screen that will bring the whole game together. This will look a little different with just a Start button that launches the first 'choices' program.

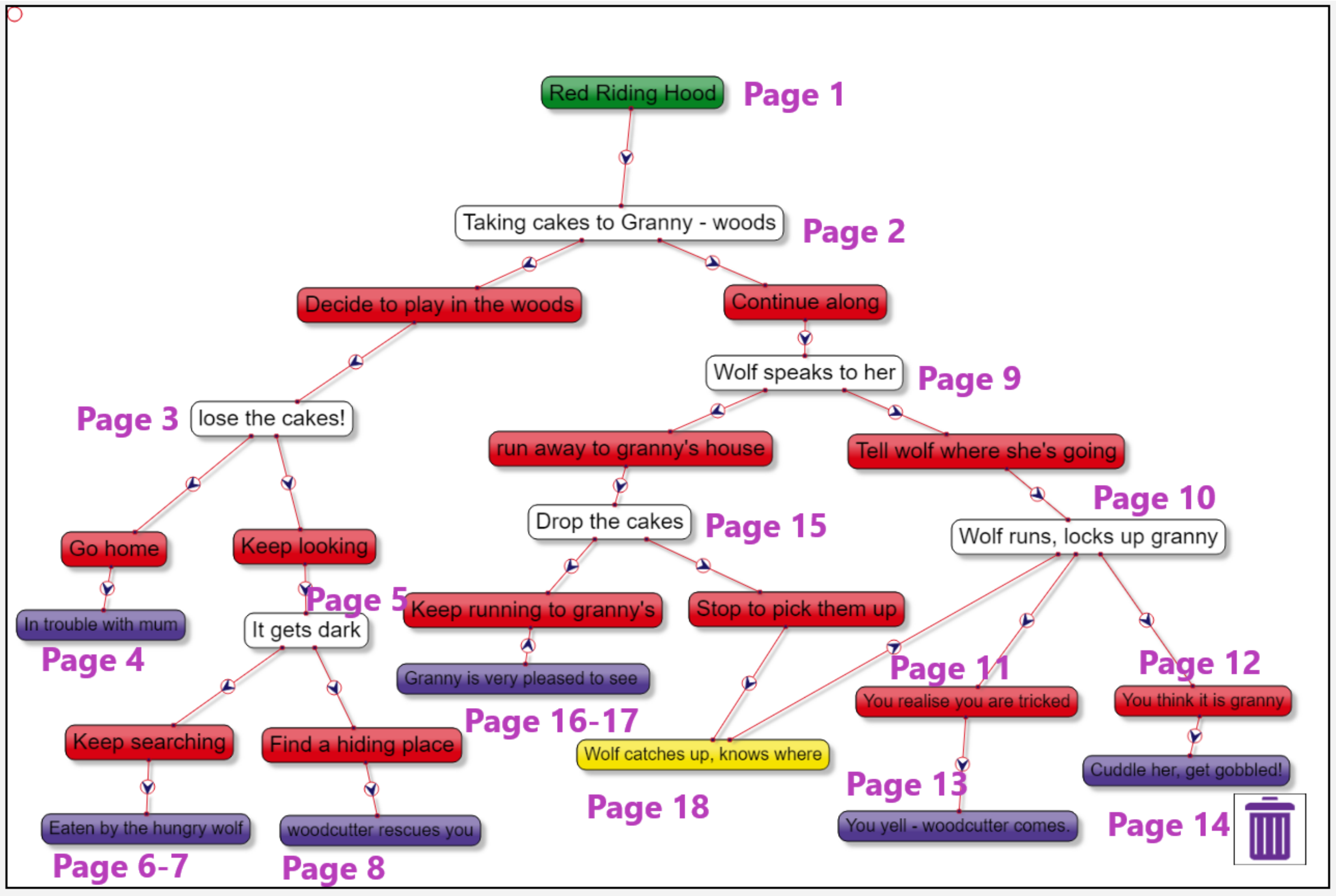
24. Once games are finished, they can be shared with others. Try sharing the games using **2Blog** to make a class blog, which other children in the school can read and comment upon. Please share your finished games with us at 2Simple as well; we'd love to see them! You can play the finished example at [Adventure In the Woods Game].

25. To **share** a game, first save it in an online folder. Then click on the 🌐 button. You have several options for sharing here, including sending an email and blogging. Click on '**Share'**, then '**Link & QR code**'. Children can click 'Copy' to copy the link and then paste it into their file by clicking in a text box and pressing the 'Ctrl' and 'V' keys on their computer.

Challenges:

Add a Restart button that appears at the end and launches the first game. TIP: You can make a text object look like a button by colouring the background. This way, the text object can be hidden until it is needed.

Appendix 5: 2Connect Diagram



Assessment Guidance

The unit overview for year 6 contains details of national curricula mapped to the Purple Mash Units. The following information is an exemplar of what a child at an expected level would be able to demonstrate when completing this unit with additional exemplars to demonstrate how this would vary for a child with emerging or exceeding achievements.

Assessment Guidance	
Emerging	Children can turn a simple story with at least one decision into a logical design using 2Connect (Unit 6.5 Lesson 1). They might need support when completing the decision tree. Children can create individual pages in 2Create a Story (Unit 6.5 Lesson 2) but will need support to link these parts in a logical way. In (Unit 6.5 Lesson 3), they can design a simple map with a sequence of rooms and one item to collect. In (Unit 6.5 Lesson 4), they will need support to turn their designs into code but can succeed in representing the player navigating to different rooms. They can debug a simple program with support. In (Unit 6.5 Lesson 4), they will need support to relate the examples to their own design, especially when using variables, but will be able to code some of the elements of their own design independently and can write code that take input from the user. Children can relate the example design to the example program and can predict what will happen in the program using the design document. In (Unit 6.5 Lesson 4), they can use their design to test whether their program has bugs but will need support to identify where these bugs are in their code and to fix them.
Expected	Children can turn a simple story with 2 or 3 levels of decision making into a logical design using 2Connect (Unit 6.5 Lesson 1). Having seen an example, they can use this to make the story their own. Children can create the pages for the component parts of the design in 2Create a Story (Unit 6.5 Lesson 2) and make good attempts to link these parts in a logical way. They might need support when debugging the linked pages if things do not proceed as expected. In (Unit 6.5 Lesson 3), they can make a design map with a sequence of rooms including rooms in which the player needs to make a choice to complete the game and collect items. In (Unit 6.5 Lesson 4), they can use the example code to turn their own designs into code. Children will debug as they code and might need some support in identifying the cause of some bugs. Children can relate the example design to the example program and can predict what will happen in the program using the design document. In their own program, they can use their design algorithm to debug their adventure story. In (Unit 6.5 Lesson 4), they can use their design to test whether their program has bugs and identify where in their code, their bugs occur. Most children apply their knowledge of coding and the fundamental order of instructions through creating their own story-based adventure game. They can identify errors in their

Assessment Guidance	
	code and specifically errors that could impact on the order of events and specific actions when buttons are pressed (Unit 6.5 Lesson 2). Most children demonstrate how algorithms are useful for representing a solution to a problem e.g. During the creation of their own story-based adventure games within 2Code they can systematically test their code against its intended outcome (Unit 6.5 Lesson 2 Point 20). Most children can carefully plan before constructing digital content such as a text adventure game. Using 2Connect, children can carefully identify the data and information they need to incorporate within their intended coded games. As they advance onto coding, the children can extract and manipulate bits of data and strings of text for the purpose of their game functionality (Unit 6.5 Lessons 1 & 4).
Exceeding	Children can turn a simple story with 3 or more levels of decision making into a logical design using 2Connect (Unit 6.5 Lesson 1). They can ensure that the design is complete and logical. Children can use 2Create a Story to make the component parts of the design (Unit 6.5 Lesson 2) and link these parts in a logical way. They can then debug in a logical way using their design document if things do not proceed as expected. In (Unit 6.5 Lesson 3), they can make a comprehensive design map with a sequence of rooms including rooms in which the player needs to make a choice and collect items in a certain order to complete the game. In (Unit 6.5 Lesson 4), they can use the example code to turn their own designs into code. Children will debug as they code using their designs and notes as a guide. In (Unit 6.5 Lesson 4), they understand and can adapt the use of variables to their own design and can write code that takes input from the user and gives output to the user. Children can relate the example design to the example program and can predict what will happen in the program using the design document. In their own program, they can use their design algorithm to debug their adventure story and foresee elements that they need to code. In (Unit 6.5 Lesson 4), they can use their design to test whether their program has bugs and identify where in their code, their bugs occur. While coding, they refer to and annotate, their design with helpful notes and changes to enable them to debug and to enhance their program.