PART-I SUMMER PROJECT

Deadline:	**17:00 Friday Week 23**
Assessment Mode:	**IN LAB DEMO IN YOUR WEEK 24 LAB SESSION**
	MOODLE SUBMISSION
	REPOSITORY LINK SUBMISSION

AIMS

In this final term of your first year, there is a single task: a single cross-module project that is designed to bring together all you have learned since you joined us in October. This project represents the final piece of practical coursework for your first year and constitutes **33%** of the coursework assessment for the year. **Time to show us what you can do!**

You have a *choice* of projects to undertake from the choices below. You need to **pick just one** of these projects. Read each of them carefully before choosing which project to undertake. Please note that the projects state which programming language you **must** use for that project. These projects have been designed to build on what you have learned in SCC.111, SCC.121 and SCC.131 (where applicable). Note that some projects are unavailable / have different requirements for students not studying SCC.131 (such as those studying for Computer Science and Mathematics or Management and IT degrees).

GETTING SUPPORT

You should attend your timetabled SCC1x1 lab session **in weeks 21-23** inclusive for support, where staff and teaching assistants will be present to provide guidance, feedback and to answer your questions. **Please note that your week 24 session will be reserved for you to demonstrate the functionality of your work**. **Staff will not be able to provide support during the Week 24 lab session.**

The FAST hub will also be available daily if you need additional drop-in support. *See Moodle for more details.*

SUBMISSION AND ASSESSMENT

This work will be assessed by a member of academic staff after you follow the instructions below to submit your work. This assessment will be further informed by a practical demonstration of your work. You will be asked to present your work in your Week 24 lab session. Be prepared to take the lead in demonstrating your project, and to answer questions about it.

- You MUST submit your code to Moodle by the advertised deadline in a .zip file.
- You MUST demonstrate your work by downloading your Moodle submission and running it IN YOUR OWN WEEK 24 TIMETABLED LAB SESSION.
- You MUST share your private scc-source/GitLab repository by the advertised deadline using the version control submission instructions below.
- You MUST ensure that your project can be compiled and executed FROM THE COMMAND LINE ON A LAB PC, without using an IDE. (If the project you selected has further instructions on how to compile and run your project, you must follow them.)
- The standard University regulation on permitting late submission of coursework DOES NOT APPLY TO THIS COURSEWORK ASSIGNMENT.

FAILURE TO ADHERE TO THE ABOVE WILL RESULT IN A FAIL MARK (F4) BEING RECORDED.

VERSION CONTROL SUBMISSION

You **are expected** to use git version control for your project, and this is reflected in the marking scheme. To submit your private GitLab repository for assessment, you must do the following:

1. To generate an SSH key pair and add it to your scc-sources account Follow this guideline: `https://scc-source.lancs.ac.uk/scc-systems-team/school-documentation/-/blob/main/ssh_authentication.md`.
2. To import your keys to scc-sources following this guideline: `https://scc-source.lancs.ac.uk/scc-systems-team/school-documentation/-/blob/main/ssh_authentication.md`

3. Create an empty code repository or clone the starter repository provided by the project. **Make sure the repository is private.**
4. Grant access for Y1 academics to your repository, by following this guideline: https://scc-source.lancs.ac.uk/scc-systems-team/school-documentation/-/blob/main/sharing_project.md
5. **Submit the FULL URL of your project repository (the same link that you'd need to 'clone' it).**

MARKING SCHEME

Your work will be marked based on the following categories. Your final grade will be determined based on a weighted mean of these grades according to the weighting shown in the table below.

Marking Scheme	
Project Functionality	**70%**
- See individual project descriptions below for indicative levels of the functionality required.	
Code Structure and Elegance	**10%**
- Modularity of code	
- Use of appropriate data types and libraries	
- Use of appropriate language constructs (arrays, loops, functions, methods, classes)	
Code Style	**10%**
- Appropriate comments, code indentation	
- Appropriate name/ scope of variables and functions/methods	
Use of Git version control	**10%**
- Clean and regular commits	
- Appropriate commit messages	

In all cases a grade descriptor (A, B, C, D, F) will be used to mark your work in each category. The following sections provide an indication of the level of functionality expected.

Students may also award of a distinction (+) category overall if they feel a piece of work exhibits clearly outstanding practice. If you feel your work warrants a distinction for going significantly above and beyond the specification, **it is your responsibility to state and demonstrate this during your Week 24 lab session**.

PLAGIARISM

This project **should be entirely your own work**. In this context, that means that it is reasonable to discuss ideas with others, but you should not expect to study the code of another student, or to let them study yours. Code examples taken from the lecture notes are acceptable, as well as, **small** sections of code (a few lines, for example) from the internet - but if it is in your code, you should be able to understand and confidently explain what it does and how it works. The use of AI tools to write code on your behalf is regarded as equivalent to copying code from any external source.

USE OF AI STATEMENT

Lancaster University has adopted a Red-Amber-Green (RAG) categorisation system to guide students on their use of Generative AI (Gen AI) tools in summative assessment. This coursework is in the **RED Category; AI tools cannot be used**. Students suspected of using AI tools in assessment identified as category red, will be investigated in line with institutional academic integrity procedures. For further information, please refer to the AI Policy. If you are unsure about the use of AI tools in your assessment, please consult your lab academic.

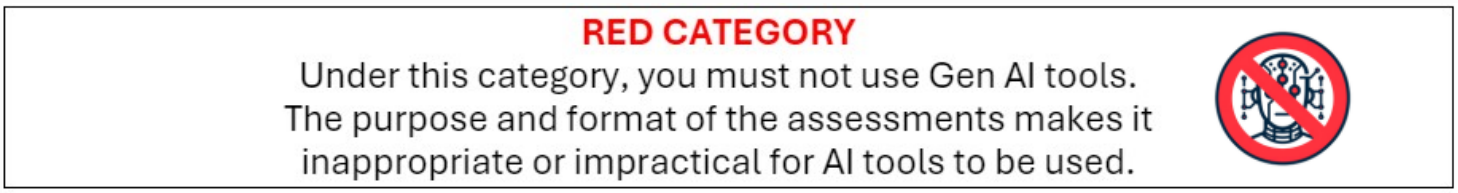


Figure 1: RED Category; AI tools cannot be used

PROJECT 1: CACHE NOISETTES GAME

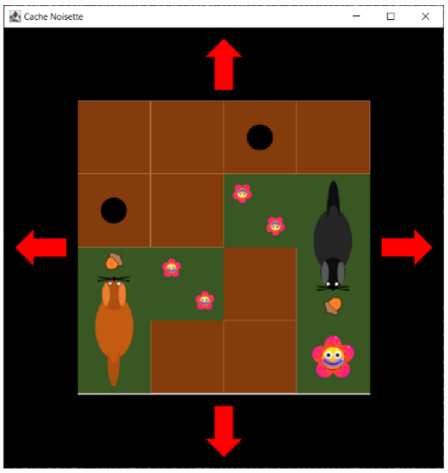
(LANGUAGE: JAVA)

PROJECT OVERVIEW

Your assignment is to create a Java Swing based interactive puzzle game called **Cache Noisettes** using Java and Swing.

I bought this game on holiday in France a few years ago – it roughly translates as "Hide the Nuts". It's a single player puzzle game that involves placing a number of squirrel pieces on a 4x4 grid, and sliding them (following specific rules) such that they each drops the nut they are carrying into a hole in the gameboard. Sound easy? Not so much… the squirrel pieces have a nasty habit of covering the holes…!

A photo of the Cache Noisettes game is shown below, alongside an example of what your Java version might ultimately look like (although you are free to design your solution as you wish):



GAME RULES

The rules of the game are deceptively simple!

- The game board consists of a 4x4 grid of empty spaces.
- There are holes in four of those spaces. These are always in the same place, as illustrated:

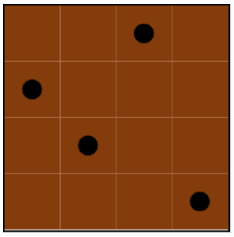


Figure 2: An empty cache noisette board

- There are game pieces that look like squirrels. *Up to four* squirrel pieces may be placed on the board, each has its own colour and shape. At the start of the game, each squirrel carries a nut. The nut resides on the same square as the squirrel's head.
- There are many levels to the game (*sixty* levels in the full game!). Each level defines which of the squirrel pieces are in use for that level, their starting position on the board, and the rotation of the pieces (a squirrel may face up, down, left or right on the board). Once placed at the start of the game, the rotation of the pieces *cannot be changed*.
- There is also a game piece that looks like a flower, that may be placed into one of the holes, which *makes that hole unusable*, and also means that no other game piece may occupy that space on the board. The position of the flower (if any) is also defined for each level.
- Once the game starts, the player may move any of the squirrel pieces left, right, up or down on the board, one space at a time. Squirrels may not move diagonally, and no part of a squirrel may leave the 4x4 grid. Similarly, no part of a squirrel may be moved into a space that is occupied by another piece (such as another squirrel or a flower). Any part of a squirrel may however move onto a hole.
- If the part of a squirrel holding a nut moves over a hole, the nut drops into that hole. A hole may only hold at most one nut and *cannot* be removed.
- The game is won when every nut is placed into a hole.

RESOURCES

Since SCC.111 is a Software Development course and not a course in computer graphics, we've provided some images for you and a class **Picture** which loads an image and can rotate the graphics for you. Clone this as the starting point of your game:

```
git clone https://scc-source.lancs.ac.uk/scc.Y1/scc.111/cachenoisettes-dist.git
```

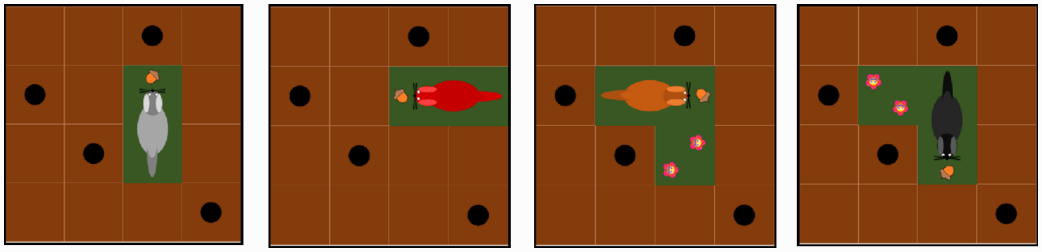
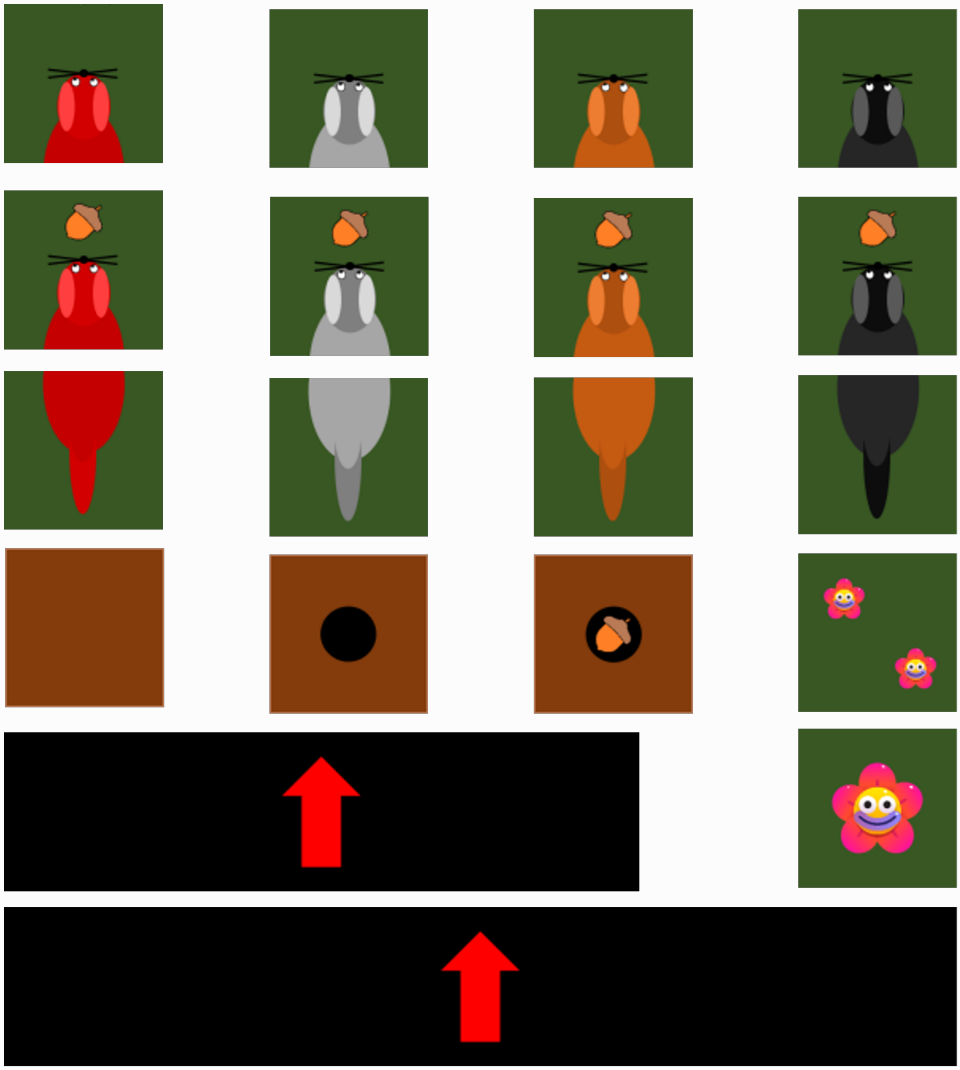


Figure 3: The squirrels in the game, facing up, left, right and down respectively. Note the Grey and Red squirrels are straight and occupy two board spaces, but the Brown and Black squirrel pieces are 'L' shaped and occupy three spaces.

Feel free to make use of these images in your program. However, if you prefer to create your own graphics for fun, this is of course just fine.



TASKS

1 - Creating the Board…

- As professional software developers, you know you should be using a version control system for your project… So start by creating a **private** gitlab repository for your work. :smiley:
- Download the image resources by **cloning** the distribution repo and extract them into the directory where you will be developing your code.
- Create a class to represent your game board, including a Graphical User Interface (GUI) using Swing. **Think carefully about which Swing classes will help you here.** Refer back to the lectures if you need a reminder on how to do this.
- Remember to write a constructor for your class that uses Swing to show a window with an empty game board.
- You are free to develop your work using your own computers, **but the code you write must work on the SCC Ubuntu virtual machine on mylab.lancs.ac.uk**. This is where your work will be marked.
- Similarly, we recommend you use VS Code and the command line to write and compile you code. If you choose to use something else, **ensure your submitted work can be compiled and run from the command line using a simple javac command**. Failure to abide by these rules may result in a fail mark for the functionality part of your work.

HINT: Images in Swing applications are represented by a class (of course!). We have provided a class called **Picture** for you to use in your solution. The **Picture** class contains a constructor that lets you create an instance of a **Picture** from a given filename and chosen rotation (in degrees).

The **JButton** class is able to create buttons showing images as well as text. In particular, the **JButton** class contains a constructor that allows a **JButton** to be created from a **Picture** as follows:

```
Picture p = new Picture("Empty.png", 0);
JButton b = new JButton(p);
```

2 - Adding pieces…

- Update your program so that it can create the Cache Noisettes Level 1, as shown below. Note that you need only support the red and grey squirrels at this stage.

- Pay particular attention to how you choose to represent squirrels in the game. For example, you may wish to consider creating a class to represent a **Squirrel** to help keep your code simpler and more elegant.
- When using the images provided, note that the filenames follow a pattern. Consider how to use this in your code to make it easy to reuse code you have written in representing different squirrels.

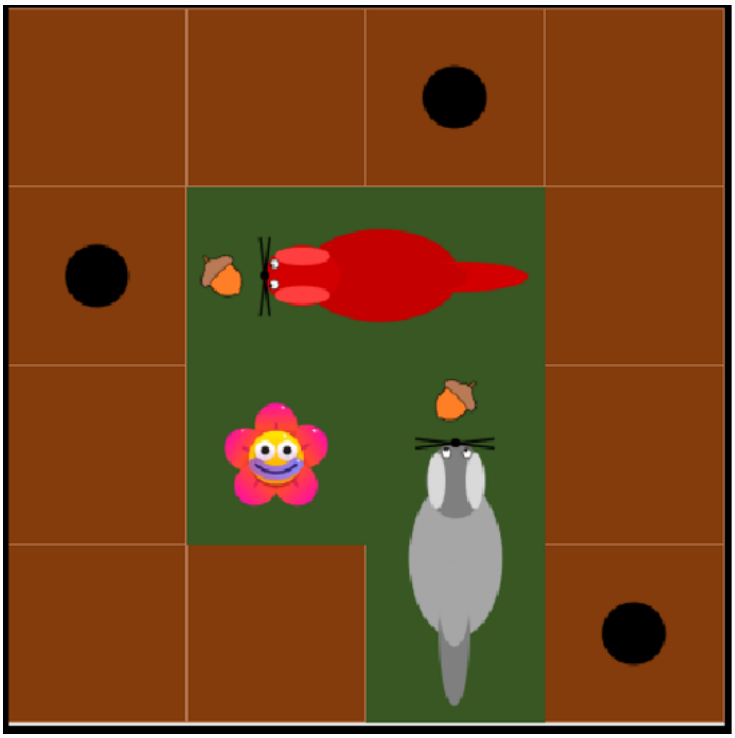


Figure 4: Figure 4: Cache Noisettes Level 1

HINT: The **JButton** class is also able to change the image shown on the button, using the `setIcon()` method. For example, a **JButton** called `b` can change what it looks like as follows:

```
Picture p = new Picture("RedSquirrel1.png", 0);
b.setIcon(p);
```

3 - Moving Pieces...

- Add buttons to your user interface to provide a way to move a squirrel around the board. An example is shown below, but you may choose any user interface you wish.
- Write a method to allow the squirrels to be moved around the board.

- You may, for example, choose to select a squirrel by clicking on its head, then use the directional buttons to move that squirrel around the board.

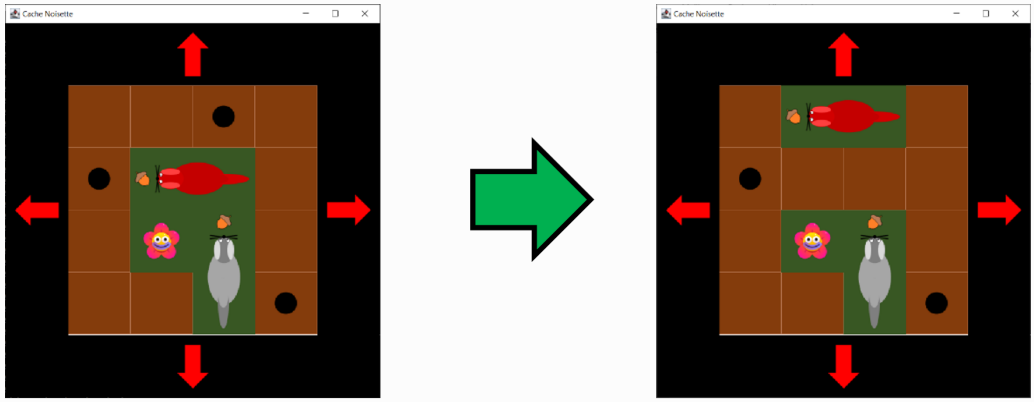


Figure 5: Figure 5: An example of the Red Squirrel being move up

HINT: Moving a **JButton** is tricky… but *changing what it displays* is much simpler. Consider how you can make use the `setIcon()` method described earlier to make it look like a squirrel is moving…

4 - Ensure Legal Moves…

- Develop your software such that it will only permit legal moves, as described at the start of this document. Any attempt to perform an illegal move should be ignored.
- Remember to verify that no part of a squirrel may leave the game board, or overlap another squirrel or flower.
- Update your software so that nuts can be placed into holes appropriately.

5 - Additional Squirrels and Levels (Intermediate Task)

Develop your program so that it can correctly support other levels.

- Note that this will require you to support the use of any of the four squirrels (which may be of a different shape), and in any orientation.
- Determine when the game is won and display an appropriate message.
- Include a simple way for users to select which level they want to play. Three levels are listed below. Images of other all 60 levels in the game are available on request in your scheduled lab session should you wish to support more in your game (although this is not expected).

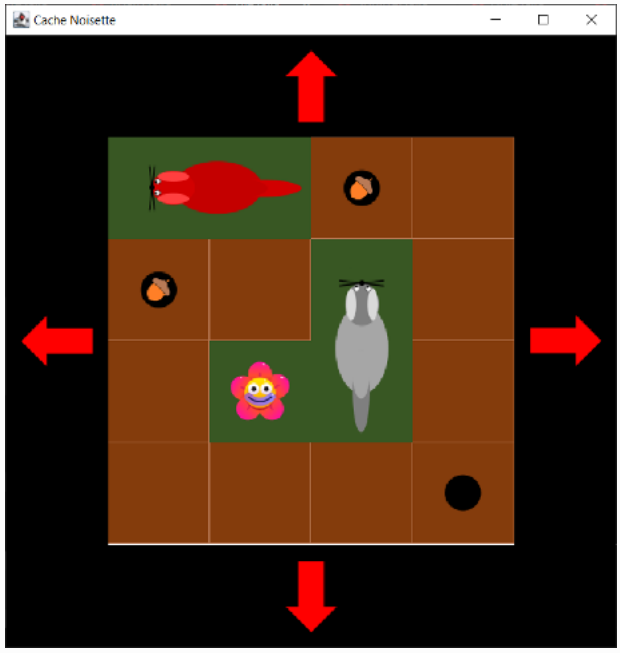


Figure 6: Figure 6: A completed game. Note that the nuts are displayed inside holes, and that the squirrels no longer carry them.

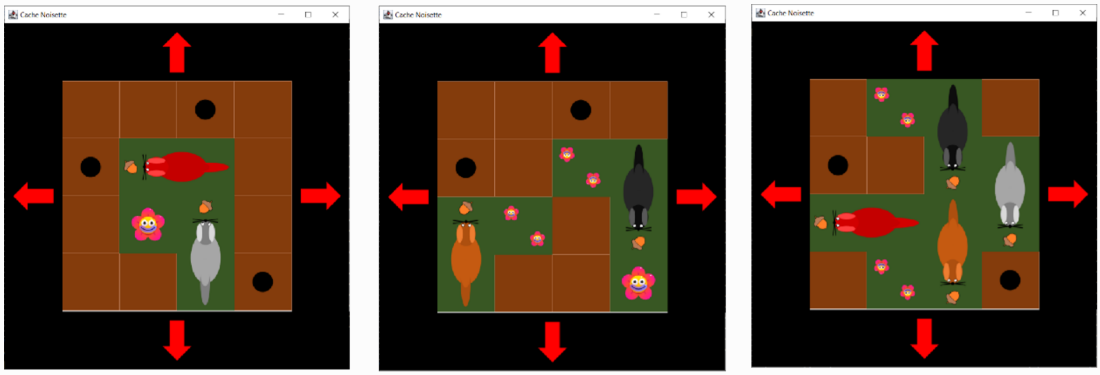


Figure 7: Figure 7: Cache Noisettes Levels 1, 10 and 40 (respectively)

Advanced task

Make it possible **to load in new levels** from a bitmap (**BMP**) file. To complete this task, you must open the file and read and intepret the data, **not** use `ImageIO` . Note that you don't have to support all of the bitmap options, you're reading the file just to detect certain patterns relating to the squares on the game board of a fixed size, so you can streamline your implementation accordingly.

Windows Bitmap (.bmp) File Format The windows bitmap format is one of the simplest ways to record image data in a file. It does not make use of compression, so its files can get quite large, but the way in which the data is stored is much simpler than other formats such as jpeg or gif to load.

Windows BMP allows for graphical data to be stored in any image width, height or colour depth (the number of bits used to represent the colour of each pixel). As such, all this information is also stored in the file itself, before the pixel data. This is very common in file formats and is known as a file header. The following diagram illustrates a simplified view of the windows bmp file. (See this explanation for a more detailed overview of the full specification for those of you that are interested!)

The left hand column shows which byte in the file is being described (`0` is the first byte, `1` is the next byte etc). The next columns describe what data is held in that location, and the final column illustrates the content using the ‘ `blankWithHoles.bmp` ’ file provided alongside this specification.

Note that many fields use multiple bytes. Such fields are all represented using unsigned, little-endian values. i.e. the least significant byte comes first, and the most significant is last. The pixel data itself uses however many bits are defined in the `BITS_PER_PIXEL` field to represent each pixel. The order of the pixels is left-to-right, bottom-to-top. Each row is also padded with zeros to ensure each row of pixels starts on a multiple of 4 bytes.

e.g. in the example below, the first `0x57` `0x7A` `0xB9` bytes indicate that the first pixel on the bottom row of the image are “colour `#B97A57` ” (you can look hex colours up here). Remember, we mentioned that the bytes are little-endian, so an RGB value appears with the blue value first. In 24 bit colour there is one byte for each of the three colours. For each line there are 48 bytes: 45 bytes (15 pixels x 3 bytes = 45 bytes) for the row in the image, then padding of 3 bytes of `0x00` to reach 48 bytes as the line must be a multiple of 4. The second line, predictably, starts with the same sequence of bytes (`0x57` `0x7A` `0xB9`) in position 102 (54 + 48). The second line is the one immediately above the bottom row.

The expanded `blankWithHoles.bmp` example image shown below, demonstrates how each tile is represented by 3x3 pixels, with a single line of white pixels separating each tile.

Each brown pixel will have the same RGB colour, the three bytes discussed previously: `0x57` `0x7A` `0xB9` The white pixels are each: `0xFF` `0xFF` `0xFF` . *Remember, the file starts bottom left and reads left to right row by row upwards.*

Byte	Name	Description	Example (blankWithHoles.bmp)
0	TYPE	Always 'B'	0x42
1		Always 'M'	0x4D
2	SIZE	Size of whole file in bytes	0x06 (0x0306 is 774 in decimal)
3			0x03
4			0x00
5			0x00
6	RESERVED	Always 0	0x00
7		Always 0	0x00
8		Always 0	0x00
9		Always 0	0x00
10	PIXEL DATA LOCATION	Offset into the file where pixel data begins	0x36 (54 in decimal)
11			0x00
12			0x00
13			0x00
14	IMAGE HEADER SIZE	Size of image header in bytes (normally 40 bytes)	0x28 (40 in decimal)
15			0x00
16			0x00
17			0x00
18	IMAGE WIDTH	Width of the image in pixels	0x0F (15 in decimal)
19			0x00
20			0x00
21			0x00
22	IMAGE HEIGHT	Height of the image in pixels	0x0F (15 in decimal)
23			0x00
24			0x00
25			0x00
26	BIT PLANES	Number of images in file (normally 1)	0x01
27			0x00
28	BITS PER PIXEL	Colour depth per pixel, Could be 1, 4, 8, 16, 24, 32	0x18 (24 in decimal)
29			
30-53	...	Not very interesting stuff :)	...
54-774	PIXEL DATA	The image data itself	0x57 0x7A 0xB9 0x57 0x7A 0xB9
			0x57 0x7A 0xB9 0xFF 0xFF 0xFF...
			0x57 0x7A 0xB9 0x57 0x7A 0xB9
			0x57 0x7A 0xB9 0xFF 0xFF 0xFF...

Figure 8: File Format of a Windows BMP file

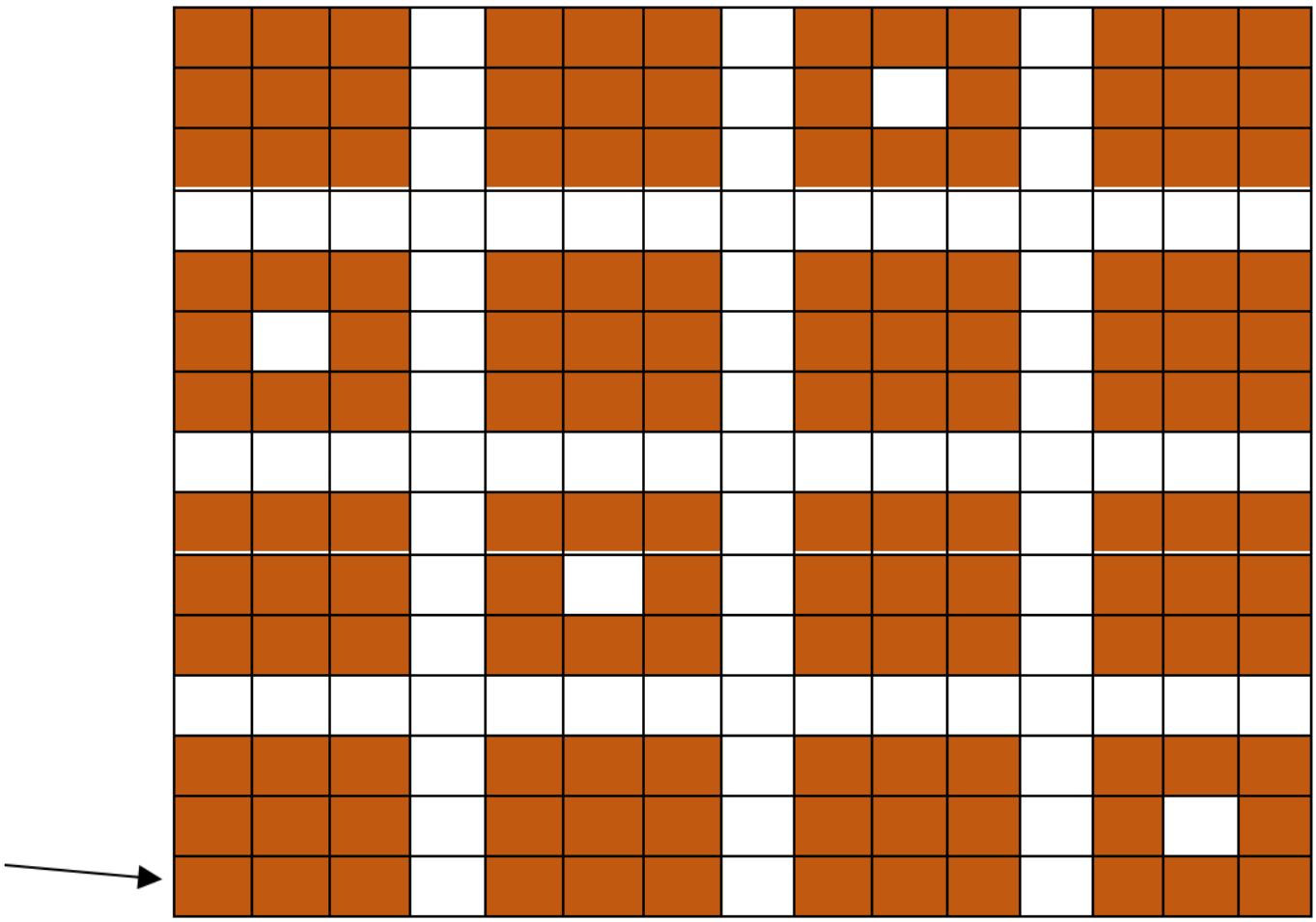


Figure 9: This diagram shows each pixel in blankWithHoles.bmp

Loading Cache Noisette Levels Once the `blankWithHoles.bmp` is loaded, it will be relatively easy to load other levels. Each of the levels will, naturally, be the same size and follow exactly the same format, the only differences will be in the RGB values stored.

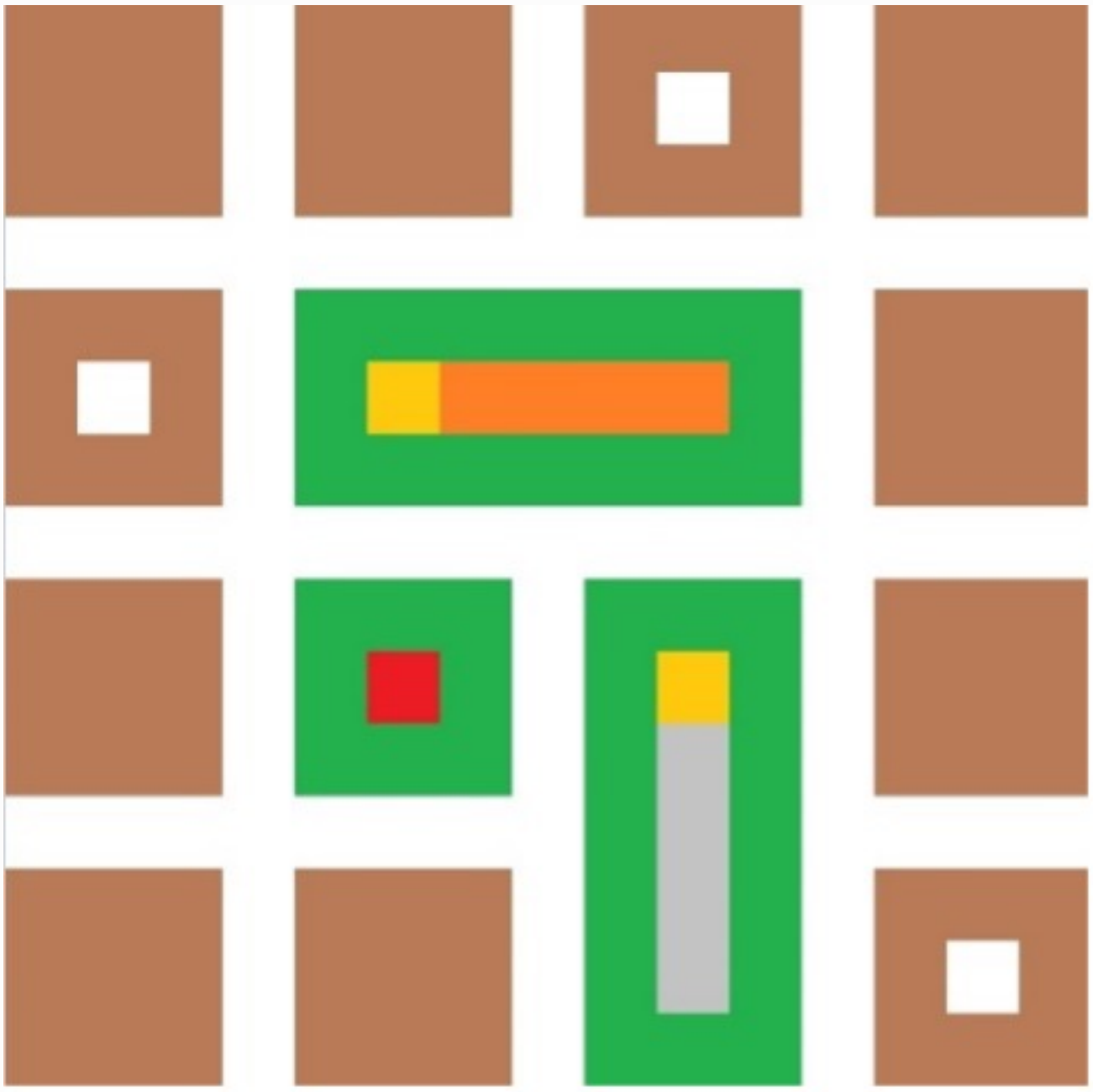


Figure 10: The bitmap for level 1.

The nut is represented by the yellow pixel in the centre of the tile. The green pieces are the squirrels, each of them with a different central colour. The single square with a red pixel at the centre represents the stationary flower.

Once level 1 is successfully loaded, and interpreted, the only further development is the final two squirrels. Level 60 – the image from the instruction book and the bitmap image alongside – using all 4 squirrels can be seen below.

Figure 11: Level 60 and it's bitmap representation.

MARKING SCHEME

Marks will be awarded according to the marking scheme shown at the start of this document. Marks for the 'Functionality' section will be awarded based on individual merit, but the following table gives an **indicative** overview of the level of functionality expected for each grade band:

SCC Major Students Studying SCC.111, SCC.121 and SCC.131

A * **Working program that meets the criteria for a A, plus:**

- Loading of levels from Windows Bitmap images or at our discretion equivalent advanced features beyond the spec.

A	**Working program that meets the criteria for a B, plus:** | | * All tasks above from 1-5, including multiple levels and | additional squirrels.
B	**Working program that meets the criteria for a C, plus:** • All tasks from 1-4, i.e. including moves and legal move detection.
C	**Working program that meets the criteria for a D, plus:** • Tasks 1-3 including correctly moving squirrels. Single level. • Board to solve is hardcoded inside an array in the program.
D	**Working program that:** • Tasks 1 & 2. Creation of level one, two squirrels, but not moving, or not moving correctly.
F	**No working program demonstrated, or program does not meet any requirements listed above.**

SCC Joint Major Students Studying SCC.111 and SCC.121 only

A*	**Working program that meets the criteria for a A, plus:** • At our discretion, advanced features beyond the spec.
A	**Working program that meets the criteria for a B, plus:** • Tasks 1-5 with reduced scope: Additional squirrel/squirrel shape handled.
B	**Working program that meets the criteria for a C, plus:** • All tasks from 1-4, i.e. pieces move and legal at least move detection for basic squirrel shape.
C	**Working program that meets the criteria for a D, plus:** • Tasks 1-3 including at least one correctly moving squirrel. • Board to solve is hardcoded inside an array in the program.

D	**Working program that:** • Tasks 1 & 2. Creation of level one, a squirrel on the board, but not moving, or not moving correctly.
F	**No working program demonstrated, or program does not meet any requirements listed above.**

PROJECT 2: JOURNEY PLANNER FOR MANCHESTER METROLINK (LANGUAGE: JAVA)

PROJECT OVERVIEW

Journey planners can be used to find the optimal route between two (or more) given locations. They are widely used to plan journeys and can be based on one, or more, modes of transport. The route may be optimised based on different criteria; for example, the journey planner may output the fastest route, the cheapest route or even the route with fewest changes depending on the user's preferences.

For this coursework, you will develop a route planner for the Manchester Metrolink based on average journey times which are provided to you in a file Metrolink_times_linecolour.csv. Each line of the file represents a single connection between two stations and gives the starting station, final station, the transport line colour and the time to travel between the stations as: From ,To ,Line ,Time (mins). You should assume that it takes the same time to travel in the opposite direction. An example of the information provided on journey times is given below. Here, for example, you can interpret the second line as it takes 6 mins to travel from Bury to Radcliffe on the yellow line and it takes 6 mins to travel from Radcliffe to Bury on the yellow line.

```
From ,To ,Line ,Time (mins)
Bury,Radcliffe,yellow,6
Brooklands,Sale,purple,1.5
Sale,Dane Road,purple,1
```

TASKS

Your task is to develop a route planner for the Manchester Metrolink. An image of the network is given below for illustrative purposes only (note, not all lines are used in the provided file, and information in the file may not exactly match the figure).

Your route planner should be based on the information and times in the provided file, not on any other timings or information. **For testing your code, we will use a new file with stations and times. The format of this file will be the same as the one given for development, but specific information on station names**

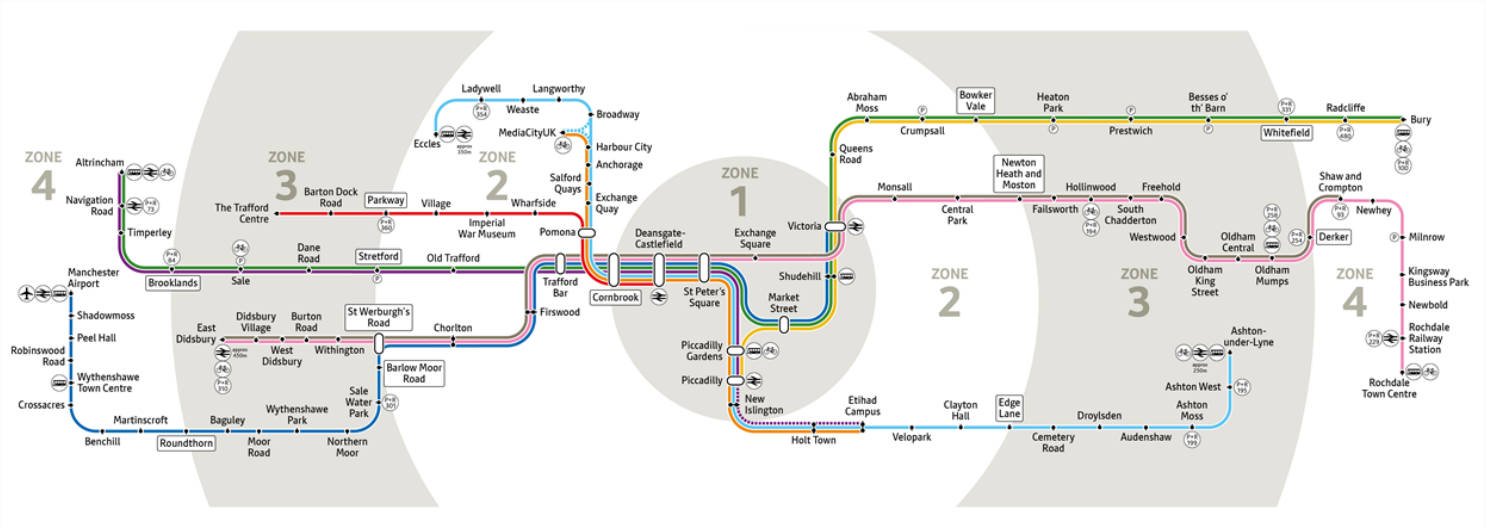


Figure 12: Manchester metrolink route map, from https://tfgm.com/ways-to-travel/tram/network-map. Map is provided for illustrative purposes only. All of your results should be based on the information given in the provided files which may not be exactly the same as this map.

and times may differ.

The program should:

- Read in information from the provided file to give station names and journey times between stations on different lines.
- Provide an interface that enables a user to specify a valid start and end station for their journey.
- Given a valid start and end station, compute a route between the two stations and output this route, including lines to travel on, journey time and changes.
- Enable users to specify constraints for optimising the journey, e.g. fewest changes, shortest time.

Full details of the expected functionality and grades are given in the task descriptions below marking.

For the provided file, an example of the shortest time journey between Salford Quays and Exchange Square, with list of all intermediate stations and any changes is given below.

```
*** Minimal Time Route ***
Salford Quays on lightblue line
Exchange Quay on lightblue line
Pomona on lightblue line
```

```
**  Change Line to red line ***
Pomona on red line
Cornbrook on red line
Deansgate/Castlefield on red line
**  Change Line to pink line ***
Deansgate/Castlefield on pink line
St. Peter's Square on pink line
Exchange Square on pink line

Overall Journey Time (mins) =  17.0
Number of Changes =  2
```

The tasks that your program must perform are detailed below. The allocation of marks for completing each of these tasks is described in the Marking Scheme.

Task 1

- Develop a user interface that enables users to input a start and end location for their journey
- Checks if user has inputted valid station names and keep asking for user to input station names until valid ones are given

Task 2

- Given any valid start and end location, come up with any valid route between the two locations
- Output the route with the list of intermediate stations Note: for a C you can make the simplifying assumption to treat everything as one single line, so you do not also have to output the line name here. To get higher than a C, you must include the different lines and times - e.g. as in Task 3. You do not need to complete Task 2 if you are able to complete Task 3.

An example of an expected output for the given file with start Victoria and end Old Trafford is given below. Note you can ouput any valid route for this task.

```
Victoria
Exchange Square
St. Peter's Square
Deansgate/Castlefield
Cornbrook
Trafford Bar
Old Trafford
```

Task 3

- Compute the shortest time route between a given start and end location. This should now include the fact there are different lines and you should make the simplifying assumption that changing trams takes 2 mins each time. Note that if multiple routes give the same shortest time, you can give any one of these routes.
- Output the route with the list of intermediate stations and lines and give the overall journey time
- **Only for SCC Major Students Studying SCC.111, SCC.121 and SCC.131**: Include explicit indication of where changes are needed when the journey is output (e.g. as in the example output given above)

An example of an expected output for the given file with start start Victoria and end Old Trafford is given below **for SCC Major Students Studying SCC.111, SCC.121 and SCC.131**

```
Victoria on pink line
Exchange Square on pink line
St. Peter's Square on pink line
**  Change Line to purple line ***
St. Peter's Square on purple line
Deansgate/Castlefield on purple line
Cornbrook on purple line
Trafford Bar on purple line
Old Trafford on purple line
Overall Journey Time (mins) =  16.5
```

An example of an expected output for the given file with start start Victoria and end Old Trafford is given below for **SCC Joint Major Students Studying SCC.111 and SCC.121 only**

```
Victoria on pink line
Exchange Square on pink line
St. Peter's Square on pink line
St. Peter's Square on purple line
Deansgate/Castlefield on purple line
Cornbrook on purple line
Trafford Bar on purple line
Old Trafford on purple line
Overall Journey Time (mins) =  16.5
```

Task 4

- Enable users to select journey constraints. Include at least the options of choosing the route with shortest time vs route with fewest changes.
- As well as the overall journey time, also give the total number of changes for the computed route and indicate where changes are needed when outputting the route.

An example of the expected output for the given file when the option to minimise changes is chosen with start Victoria and end Old Trafford is given below

```
*** Route with Fewest Changes ***
Victoria on green line
Shudehill on green line
Market Street on green line
St. Peter's Square on green line
Deansgate/Castlefield on green line
Cornbrook on green line
Trafford Bar on green line
Old Trafford on green line
```

```
Time (mins): 20.0
Total Changes: 0
```

Task 5

- **Only for SCC Major Students Studying SCC.111, SCC.121 and SCC.131:**
- The user interface for inputting start and end locations as well as constraints (e.g. shortest time vs fewest changes) should be a GUI.

GETTING STARTED

- Refresh your knowledge of graphs, as well as the algorithms on graphs that we covered in the 121 lectures
- If you aim to implement the journey planner using a graphical interface, then refresh your knowledge of the Swing classes discussed in 111 lectures
- You may find the java collections API useful. This has a set of data structures, further details are provided here https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html

MARKING SCHEME

Marks will be awarded according to the marking scheme shown at the start of this document. Marks for the 'Functionality' section will be awarded based on individual merit, but the following table gives an **indicative** overview of the level of functionality expected for each grade band:

SCC Major Students Studying SCC.111, SCC.121 and SCC.131

A*	**Working program that meets the criteria for a A, plus:** • Meets the requirements of Tasks described under Star Award
A	**Working program that meets the criteria for a B, plus:** • Meets the requirements of Tasks 4 and 5

B	**Working program that meets the criteria for a D, plus:** • Meets the requirements of Tasks 3
C	**Working program that meets the criteria for a D, plus:** • Meets the requirements of Task 2
D	**Working program that:** • Meets the requirements of Task 1
F	**No working program demonstrated, or program does not meet any requirements listed above**

SCC Joint Major Students Studying SCC.111 and SCC.121 only

A*	**Working program that meets the criteria for a A, plus:** • Meets the requirements of Tasks described under Star Award
A	**Working program that meets the criteria for a B, plus:** • Meets the requirements of Task 4
B	**Working program that meets the criteria for a D, plus:** • Meets the partial (as in description) requirements of Tasks 3
C	**Working program that meets the criteria for a D, plus:** • Meets the requirements of Task 2
D	**Working program that:** • Meets the requirements of Task 1
F	**No working program demonstrated, or program does not meet any requirements listed above**

STAR AWARD

To get an A+, in addition to the above tasks, you must include the following extra functionality (or at our discretion equivalent advanced features beyond the spec).

- As well as outputting the route as a list with changes and lines you should also produce a visualisation of the computed route. To do this you should not use any existing graph plotting libraries.
- Your journey planner should also be able to account for delays or station closures as well as also include the option of walking between stations as part of the journey.
    - You should enable a user to input details of any stations that are closed as well as specify delays on lines by inputting new times that it takes to travel between stations. You should enable any number of station closures or delays to be specified by the user. The new journeys should be computed based on this new information.
    - For the option of walking between stations you should use the times given in the provided file walktimes.csv, this file gives a matrix of the times in minutes to walk between each station. Here the indivudual station names are given in the first column and row, with the corresponding time to walk between stations given as a cell in the matrix (clearer to see if file is opened as spreadsheet, for example opened in excel). Again, to test your code we will use a new file with a new set of walking times where the times and stations may differ.
    - Your journey planner should again compute the shortest time journey and the journey with fewest changes, but this should now include the information on station closures and delays as well as the possibility of walking.

PROJECT 3: MICRO:BIT PONG (C++ AND ARM ASSEMBLY)

PROJECT OVERVIEW

For this project you must implement a single player pong game using the BBC micro:bit. The resulting project should use the built-in LED display to create a simple graphical interface for the game. The player should be able to control the paddle using the A and B buttons. Furthemore, the game should keep track of the score˙ and display it on the LED display. Your imlpementation should use C++ and the CoDAL runtime, while a small component of the coursework is expected to be implemented in ARM Assembly.

Pong is a classic video game that simulates table tennis. The single player version consists of one paddle and a ball that bounces between them. The player can move the paddles left and right, and the goal is to hit the ball with the paddle. The goal of the player is to maximize the number of time the ball bounces on the paddle. The ball can bounce on the paddle, and the top, right and left walls. If it hits the bottom wall, the game stop and the ball bounce counter has to reset. If the ball reached the bottom, and no paddle was there, then the game stops. The game should keep track of how many time the ball hit the paddle.

The primary objective is to create a pong game that:

- ☐ Offers a basic environment to play pong using the micro:bit LED display and buttons.
- ☐ Keeps track of the score and displays it on the LED display.
- ☐ Implements a basic physics models to simulate the ball's movement and collision with the paddles.
- ☐ Uses assembly to implement the physics model and improve the game's performance.
- ☐ Use advanced features of the micro:bit, such as the accelerometer and the radio, to enhance the game.

THE TASK

This project will exercise the knowledge you acquired from the modules SCC.111, SCC.121, and SCC.131. You will use C++ and assembly to complete this project and you will use the knowledge of the micro:bit CoDAL APIs. To get full marks you are expected to complete the following activities. Marks will be awarded based on the number of tasks you complete. A starting point for your coursework can be the

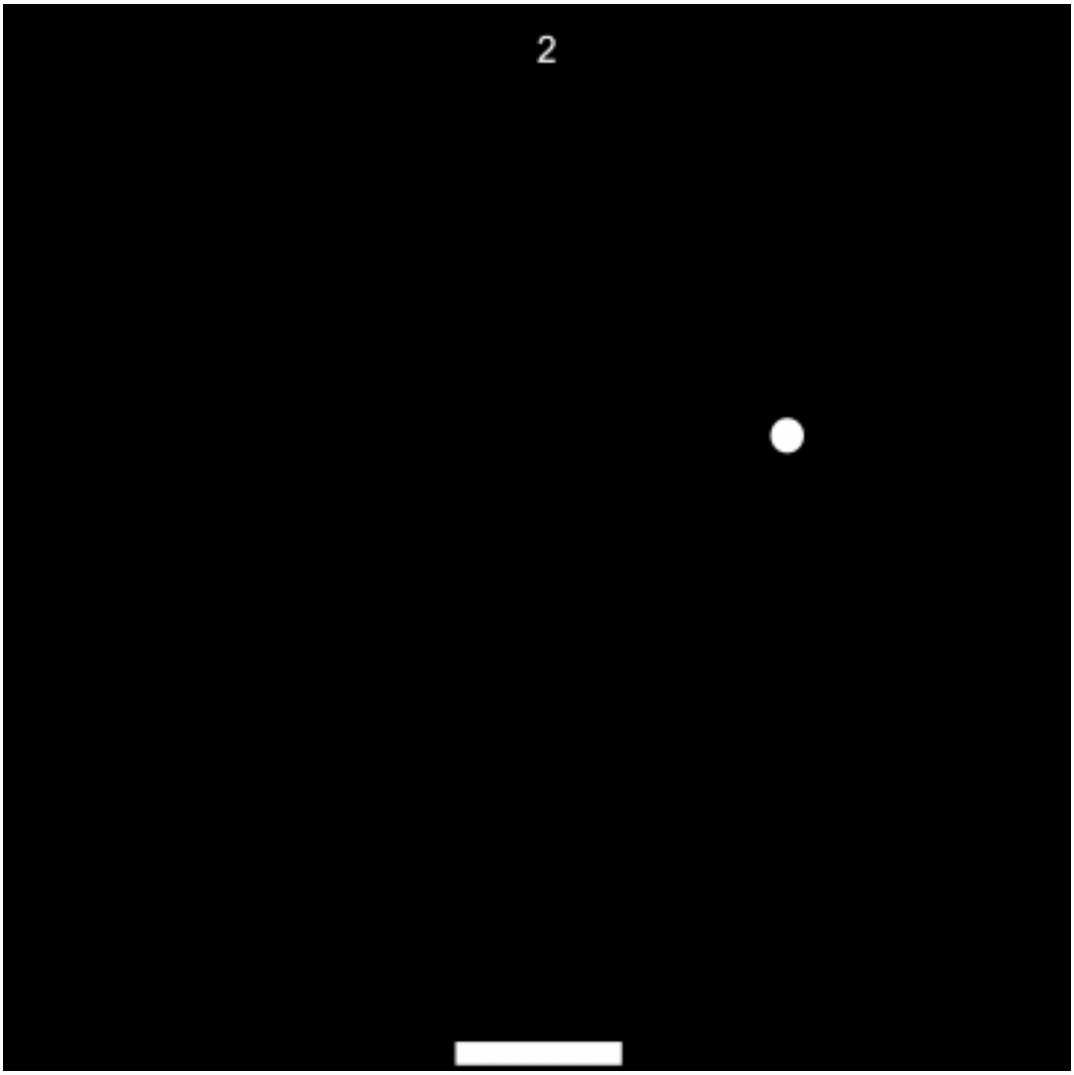


Figure 13: Single player pong game simulator

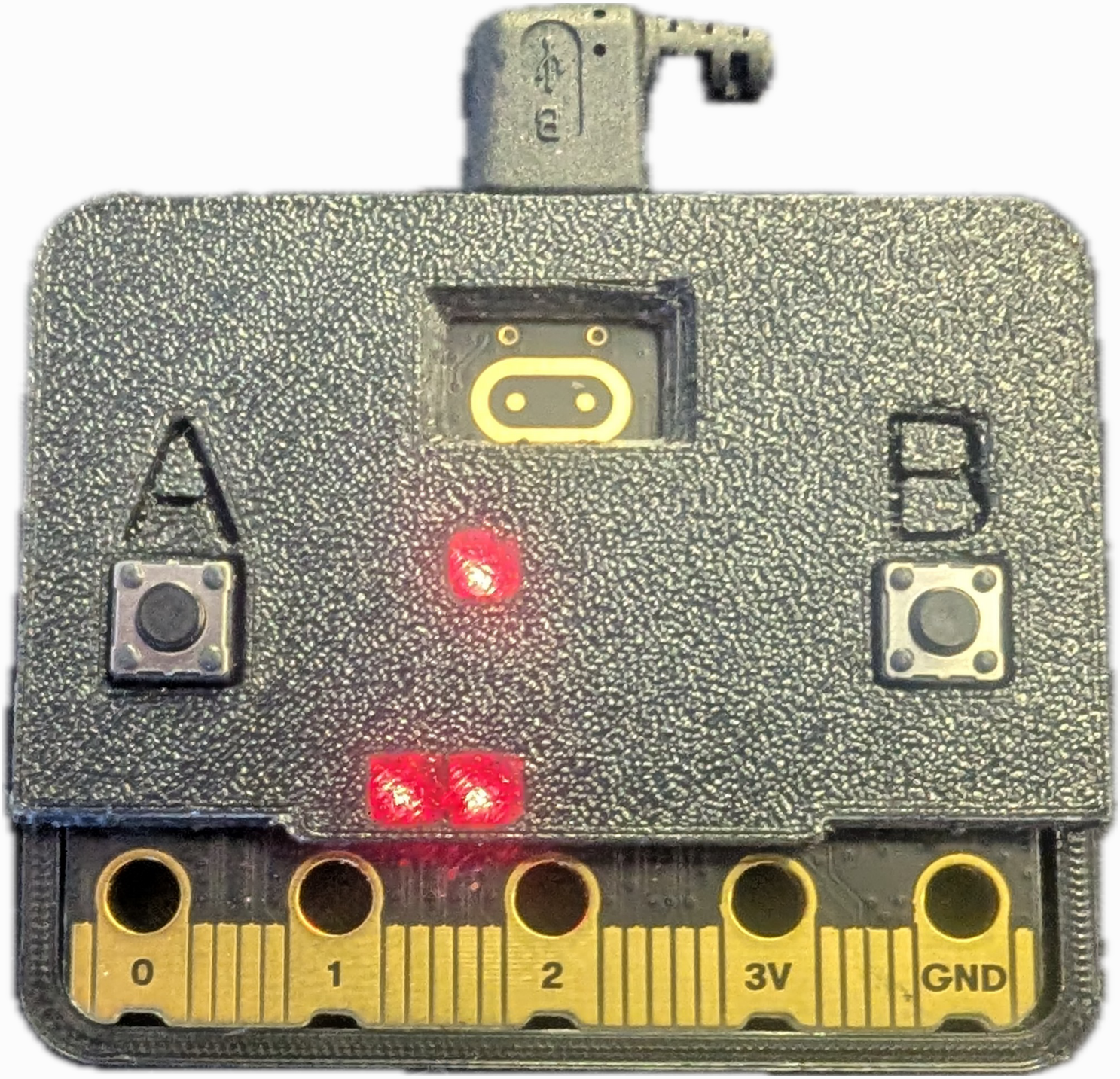


Figure 14: Pong on the micro:bit

example project at https://scc-source.lancs.ac.uk/scc.Y1/scc.131/microbit-v2-samples.

In order to implement this project, you will have to use a slightly modified pattern, from the event driven model, used in previous micro:bit projects. You should design firstly a set of global variables that store the state of the game, which can be modified by events from buttons, the network etc. The drawing of the scene should happen through a sperate thread that executed an infinite loop and perform a draw operation at fixed intervals based on the program state (e.g. ball and paddle position). This function should be implemented in a thread, a popular OS concept to model execution paralelism. A thread can run concurrently with other threads. Threads are useful for implementing tasks that need to run in the background while the main program is running. The CoDAL runtime offers a lightweight thread type, called a "fiber". You can use the following CoDAL API call to create a new thread:

```
    /**
      * Creates a new Fiber, and launches it.
      *
      * @param entry_fn The function the new Fiber will begin execution in.
      *
      * @param completion_fn The function called when the thread completes execution
↪   of entry_fn.
      *                      Defaults to release_fiber.
      *
      * @return The new Fiber, or NULL if the operation could not be completed.
      */
    Fiber *create_fiber(void (*entry_fn)(void), void (*completion_fn)(void) =
↪   release_fiber);
```

The thread should be implemented as a function with a loop, that sleeps for a fixed interval after each loop repeat. By controlling the sleep duration, you can control how fast or slow the game runs. The thread should be created in the main function using the CoDAL fiber API. The following code snippet shows how to create a new thread and the function that will be executed by the thread:

```
    int main() {
        ....
```

```
    create_fiber(draw_scene);
    ....

    release_fiber();
}
// Recompute paddle and ball position and update LED display
void draw_scene() {
    while (1) {
        // TODO Add code
        fiber_sleep(1000);
    }
}
```

We organized the implementation of the pong game in four tasks. You should complete each task in order to implement the full game.

Task 1: Display the game environment

For this first task, you are expected to use the CoDAL runtime to display the basic game environment on the micro:bit LED display. Your logic should be implemented as part of the `draw_scene` function.

The game environment should include the following elements:

- A **fixed** pong ball display in position (0, 0) of the display.
- A paddle placed at the bottom of the display with a width of 2 pixels.

Task 2: Implement the game logic

For this second task, you are expected to implement the logic that will allow the code to to move the paddle across the bottom of the screen. Your ball can be fixed when implementing this task.

The game logic should include:

- Event handlers for the A and B buttons should update the global state of the paddle position.

- Code in the `draw_scene` function that updates the paddle position based on the global state, everytime the LED display is redrawn.

Task 3: Ball movement and game logic

For this third task, you are expected to implement the logic that will allow the ball to move across the screen and bounce off the walls and the paddle. The ball should move in a straight line. The ball should bounce off the walls and the paddle when it collides with them. Ball movement computation should be part of the `draw_scene` function, but can use a separate function in order to make Task 4 easier. As an example you can use the following function signature:

```
int update_ball_position(int ball[], int ball_direction[], int paddle[], int *score);
```

Each of the input parameters should be a pointer to an array of integers. The `ball` position should be a 2D array with the x and y coordinates of the ball. The `ball_direction` should be a 2D array with the x and y direction of the ball. The `paddle` should be a 2D array with the x and y coordinates of the paddle. The `score` is a pointer to the variable that stores the score. Finally, the function should return an integer that indicates if the game has ended because the ball hit the bottom wall and the paddle did not deglect it. You implementation can assume a fixed panel size of 5x5 pixels.

In order to implement the ball movement, you should consider a way to model the ball movement. A good starting point to implement the logic is to seperate the ball state into two components: the position and the direction. The ball position stores the x and y coordinates of the ball. The direction should stores the direction of the ball in the x and y axis. A value of 1 could represent a downward movement in the x axis, or to the right in the x axis. Similarly, a value of -1 could represent an upwards movement in the y axis, and to the left in x axis. When a ball bounces off a wall, the direction should be inverted. For example, if the ball is moving downwards and collides with the top wall(i.e., ball y is 0), the direction should be inverted to upwards. Similarly, if the ball is moving downwards and collides with the paddle (i.e., ball y is 3), the direction should be inverted to upwards. The ball should also bounce off the left and right walls, but the direction should be inverted in the x axis.

The game logic should include:

- Implement code that updates the ball position based on the direction of the ball.
- Implement code that checks for collisions with the walls and the paddle.

- KEeep track of the score (how many times the paddle hit the ball) and display it on the LED display.

Task 4: Ball movement in ARM assembly

For this last task, you are expected to implement the ball movement logic in ARM assembly. The assembly code should be called from the `draw_scene` function and should update the ball position based on the direction of the ball. A good starting to design your function is the function `update_ball_position` in Task 3. The assembly code should follow all the functions conventions that we discussed in the relevant lectures. In order to allow the CoDAL build system to find your assembly code, you should place the assembly code in a file called `pong.S` and place it in the same folder as the `main.cpp` file (i.e., `source/`). The assembly code should update the ball position array and compute if the game has ended. In order to make your function visible to the C++ code, you should use the `extern "C"` keyword in your C++ code. Foe example if the the assembly function is called `update_ball_position_assebly` , you should declare the function in your C++ code as follows:

```
extern "C" int update_ball_position_assembly(int ball[], int ball_direction[], int
↪  paddle[], int *score);
```

Similarly, the assembly code should declare the function as follows:

```
.global update_ball_position_assembly
update_ball_position_assembly:
    // TODO Add code
    bx lr
```

MARKING SCHEME

Marks will be awarded according to the marking scheme shown at the start of this document. Marks for the 'Functionality' section will be awarded based on individual merit, but the following table gives an **indicative** overview of the level of functionality expected for each grade band:

A	**Working program that meets the criteria for a B, plus:** • Meets the requirements of Task 4
B	**Working program that meets the criteria for a C, plus:** • Meets the requirements of Task 3
C	**Working program that meets the criteria for a D, plus:** • Meets the requirements of Task 2
D	**Working program that:** • Meets the requirements of Task 1
F	**No working program demonstrated, or program does not meet any requirements listed above**

The precision of step counting will not be assessed, so do not worry if you step counting is a bit off. Our focus will be to ensure that you implement the described functionalities.

STAR AWARD

An A* is also available, for students wishing to really push their limits! We will award an A* for a project that is of exceptional quality, and demonstrates a level of understanding and creativity that goes above and beyond the requirements of the project. This could be through the implementation of additional features. A few ideas include:

- Implementing a two-player mode using the micro:bit radio.
- Use the on-board accelerometer to control the paddle.