,

NEA Computer Science

Daniel Vousden

Barton Court Grammar School

Center Number: 61403

Contents

[Part A 6](#_Toc99025946)

[About My Project 6](#_Toc99025947)

[Stakeholder 6](#_Toc99025948)

[Research 6](#_Toc99025949)

[Features 8](#_Toc99025950)

[Limitations 8](#_Toc99025951)

[Hardware 8](#_Toc99025952)

[Success Criteria 8](#_Toc99025953)

[Part B 10](#_Toc99025954)

[Decomposition 10](#_Toc99025955)

[UML Diagram 11](#_Toc99025956)

[Hierarchy Diagram 11](#_Toc99025957)

[Flowchart 13](#_Toc99025958)

[Usability Features 16](#_Toc99025959)

[Key Variables 17](#_Toc99025960)

[Testing Data 20](#_Toc99025961)

[Post Development Stage 23](#_Toc99025962)

[Part C 25](#_Toc99025963)

[Stage 1 25](#_Toc99025964)

[Stage 2 25](#_Toc99025965)

[Stage 3 35](#_Toc99025966)

[Stage 4 37](#_Toc99025967)

[Stage 5 50](#_Toc99025968)

[Stage 6 73](#_Toc99025969)

[Stage 7 104](#_Toc99025970)

[Stage 8 112](#_Toc99025971)

[Part D 125](#_Toc99025972)

[Post Development Testing 125](#_Toc99025973)

[Usability Testing 135](#_Toc99025974)

[Success Criteria 141](#_Toc99025975)

[Limitations and Maintenance 145](#_Toc99025976)

[Further Development 146](#_Toc99025977)

# Part A

## About My Project

My program is a game in which the user must navigate a maze collecting items allowing them to progress through levels while being challenged by a different range of enemies which come with different difficulties depending on the user’s choice. The enemies will navigate the maze using an AI which makes decisions based on the user’s movements and each level will be randomly generated also influencing the AI’s decisions. This is done to make sure the game remains fresh and interesting for the user as each randomly generated level and enemy difficulty will bring a new experience for the user.

## Stakeholder

|  |  |  |
| --- | --- | --- |
| Stakeholder | How They Will Use My Product/ Affected by It? | How Is My Product Appropriate? |
| Young Children | Young children can use this product to pass the time and develop problem-solving skills while enjoying the in-game objectives that have to be completed. | My product is appropriate for younger children as the objectives in the game are easy to understand, and difficulty can be adjusted by the user. |
| Parents | This product will affect adults as their children may choose to play the game/product. This involves them as they will have to make sure the child’s game is appropriate for their age group. | My product is appropriate as it has a simple appearance with no inappropriate content. However, if a parent wants to check, the UI will be easy to navigate and check for their child. |
| Casual Gamers | Casual gamers use this product game for enjoyment purposes and as a way to challenge themselves with varying difficulty. | My product is appropriate for casual gamers as it is simple to pick up and play as the rules and objectives are simple, and the difficulty can be changed to suit the player’s skill level. |
| Competitive Gamers | Competitive gamers would use this product to challenge themselves by attempting to achieve a new high score each game. | My product is appropriate for competitive gamers due to the game’s simple objectives yet complex levels. In order to achieve a higher score the user will have to strategically plan their moves adding a competitive aspect to the game. |

## Research

Overview

Pac Man eChalk

The Pac Man game featured on the eChalk’s website is a small, lightweight minigame that game be accessed and played on the web. The game features 2d graphics and a simple maze the user must navigate, collecting pellets as they progress while being chased by ghosts. The minigame also has a life system allowing the user to have multiple attempts at the same level if required and features a single map with a score counter.

Features I can apply to my project

My solution will also feature 2d graphics, although this is more of a coincidence than something inspired by Pac Man due to my time restrictions. The scoring system was very effective and gave me a purpose while navigating the maze making the game more interesting, which I found to be a crucial feature. Also, the implementation of a life system that I found increased playtime because I could carry on with my previous progress instead of having to restart. Implementing a score counter, and life system to my solution will benefit it immensely as it could potentially increase user interest.

A picture containing text

Description automatically generated

<https://www.echalk.co.uk/amusements/Games/Pacman/pacman.html>

Maze (Steinbach)

The maze game featured on Steinbach’s website is a minigame in which the user has to navigate increasingly tricky mazes depending on the mode chosen. The user must start at a given position in a maze and then choose which path they would like to follow each time a junction is reached. The game uses simple 2d graphics and keeps track of your progress (which level you are on).

Features I can apply to my project

My solution will also feature multiple levels as I believe this would make the game more interesting for the user however I feel that set levels may become repetitive due to the fact that more dedicated players may complete the given levels and the game will require further development. I feel that a random level generator would be more effective for my solution as this would mean once the game is developed the product will not require any updates to add extra levels. I also appreciate the simplicity of the menu within the game which consists of a home button, a cross, and an audio button. I think a simple menu will be suitable for my game as it will be easy to navigate the options the game will provide.

Qr code

Description automatically generated

<https://www.mysteinbach.ca/game-zone/1507/maze/>

## Features

GUI- A GUI makes my game more interactive for the user and overcomes some of the limitations set by not having a GUI.

Scoring System- This system will keep track of a score assigned to each objective/item completed. This will provide the user with a sense of progression while playing the game making it more enjoyable.

Difficulty Setting- This would allow the user to alter the difficulty of the game making the game readily available to the users of a range of skill levels.

Different Levels- This will keep the game fresh and replay able for the users as repeating the same level would become boring and repetitive so a range of different levels will increase playtime and playability.

Enemy AI- The AI will evolve/ change with the difficulty setting which will keep the game more interesting for the user as they can challenge themselves and it provides a sense of improvement.

Random Maze generation- The levels would be randomly generated using an algorithm that would make each level different meaning the user cannot repeat the same level over and over meaning the game will not have to be updated with extra levels in the future.

## Limitations

The first main limitation of my solution includes the creation of more complex graphics (3D) as this would require too much time to learn and apply to my program. Another limitation includes the fact that the levels are randomly generated as although this provides more diversity to the game it may cause some levels to become naturally harder than others meaning a more complex algorithm would be required to generate the mazes with certain parameters however this would make the levels less random.

## Hardware

The hardware required to run/use my program will include a standard PC which has a java virtual machine on it, a mouse, and a keyboard. This hardware will be required as my program will be programmed in java and would require the virtual machine to operate and a mouse and keyboard would be required to interact with my game.

## Success Criteria

|  |  |  |
| --- | --- | --- |
|  | Target | Relevance |
| 1 | The levels on the game will be randomly generated | The levels being randomly generated adds a certain complexity to the game as the user will have to adapt to a new scenario at each level and it Is crucial the user has levels that they have not done before as repetition can become boring. |
| 2 | There will be a score counter on my game | This will be implemented to allow the user to monitor their progress within the game as It may be hard to play without a sense of progression. Meaning the user will become more invested in the game. |
| 3 | There will be different difficulty settings | This will be implemented as it will challenge users of different skill levels against a suitable AI matching the skill level chosen. |
| 4 | The user can move through the maze | This is relevant as the user will not be able to play/ complete the game without an input that manipulates the in-game player’s movement. |
| 5 | The enemies in my game will navigate the maze towards the user. | The enemies not moving randomly is important as the enemies thinking for themselves and moving strategically will challenge the player more and make the game harder. |
| 6 | There will be a timer within the game | This will be implemented to allow the user to monitor their progress within the game as it may be hard to play without a sense of progression. Meaning the user will become more invested in the game. For example, they may strive to achieve a better time the next time they play. |

# Part B

## Decomposition

|  |  |  |
| --- | --- | --- |
| Sub-process | Explanation | Justification |
| Title Menu UI | The Title Menu UI is a sub process due to it being a large part of my project and a main component of the game. Having this as a sub process will also make it easier to program. | After each game the user will have to navigate back to the menu screen meaning that as a sub process the menu ai can be easily called and reused within my program. |
| In Game UI | The in-Game UI is a sub process due to it being a large part of my project and a main component of the game. Having this as a sub process will also make it easier to program. | Each time a game is initiated the UI will have to be changed from the menu UI to the in-game UI and due to the repetition, it will save time and space as a subprocess which can be reused. |
| Maze Generation | The Maze Generation is a sub process due to it being a large part of my project and a main component of the game. Having this as a sub process will also make it easier to program. | The generation of a random maze is quite complex so having a larger sub process would make the task too complicated. The sub process could also be reused with ease throughout the program. |
| Enemy AI | The Enemy AI is a sub process due to it being a large part of my project and a main component of the game. Having this as a sub process will also make it easier to program. | The enemy AI will be relevant during a large portion of the experience using my program making it suitable to be a subprocess so it can be easily reused. |
| Score Counter | The Score Counter is a sub process due to it being a large part of my project and a main component of the game. Having this as a sub process will also make it easier to program. | The score counter is the most separate part of my program meaning it only interacts with the interactions faced by the player meaning it should be a sub process to avoid unneeded complexity added while coding. |

## UML Diagram

Diagram, schematic

Description automatically generated

## Timeline Description automatically generatedHierarchy DiagramDiagram Description automatically generated

## Diagram Description automatically generatedFlowchartChart, diagram Description automatically generatedDiagram Description automatically generatedDiagram Description automatically generated

## Usability Features

|  |  |  |
| --- | --- | --- |
| Usability Feature | Description | Justification |
| Buttons | Standard buttons that can be pressed by the user and used to navigate the menu. | The use of buttons is to reduce the ability for the user to input the wrong data format or incorrect data meaning there will be less chance of the program failing to respond to the user’s input. Buttons also reduce confusion for the user as no complex input is required. |
| GUI | A GUI to hold/ display the parts of my game. | The use of a GUI is necessary for my game as without the GUI the user will not be able to navigate any part of my game/ it will be very difficult. |
| Font | A large bold clear font | This adds to usability due to a large, bold font being easy to read and therefore making the game easier to navigate. |
| Images | Clear simple images representing parts of the menu and in game display. | Clear simple images will help the user understand and use the menu/ in game display. An example is the End Game screen being an image, this will convey clearly to the user the game has ended instead of causing confusion with small text. |
| Layout | Images, subtitles, and buttons will be placed on complementary colours and together with the relevant information. | The layout including complementary colours will cause buttons, subtitles, and images to remain clear and easy to view. The layout also includes the exit and minimise buttons to be placed in the top right of the UI like many other programs as this will be familiar to the user. |
| Colours | Colours will be matched suitably to maintain an easy to view UI and text will remain easy to read when placed on these colours. | Complementary colours will add to the appearance of the UI and make it easy for the user to read information from the display. |
| Instructions | Standard ordered instructions that will explain the game to the user. | These instructions will reduce confusion and will leave the user with very little to have to figure out on their own. This support will allow the user to swiftly enjoy the game. |

## Key Variables

|  |  |  |
| --- | --- | --- |
| Key Variables, Data Structures and Classes | Justification | Validation |
| Score - long | The Score is required to store the score throughout the game. The score must be easily added to which can be done easily with a long variable. The long variable can store up to 9,223,372,036,854,775,807 and with each item being 100 score it is unrealistic that the user will collect enough items before the counter resets. | The program will check that the user does not exceed the highest score possible if it is reached (9,223,372,036,854,775,807). Although this is unlikely it may cause the score counter to display incorrect values if exceeded. |
| mazeLayout - int [][] | The mazeLayout 2D Array is required to store the layout of the maze and make the maze easy to change as the program progresses. | There is no validation required for mazeLayout as it will use predefined values which cannot be changed by the user. |
| Enemy - obj | The enemy array will be used to represent and hold how many enemies there are and be easily accessible when enemy locations and such are required. | The program will check that the object is not in an illegal location. For example, outside of the board or on a wall tile. This can be done by verifying the xLocation and yLocation and comparing them to the maze array. |
| Timer - int | The Timer variable will be used to hold how many seconds the user has survived for. And will be displayed for the user to see. | The Timer variable will be checked to make sure it does not exceed 2,000,000,000 seconds because as mentioned earlier if this number is exceeded the number displayed may be incorrect. |
| Dimensions - int | The dimensions variable will be used to declare the dimensions of an object within the program. | This does not require validation as the dimensions of the tiles will be pre-determined and not accessible to the user to change. |
| xLocation - int | xLocation will hold the X coordinate of any object within the maze or the array column more specifically. | The xLocation variable will be validated by checking the xLocation is not in a location it is not supposed to be. It will check the xLocation is within the board dimensions and won’t appear on a wall tile box. |
| yLocation - int | The yLocation variable will hold the Y coordinate of any object within the maze or the array row more specifically. | The yLocation variable will be validated by checking the yLocation is not in a location it is not supposed to be. It will check the yLocation is within the board dimensions and won’t appear on a wall tile box. |
| Difficulty - int | This variable is declared to hold the difficulty the user has chosen and will be referred to when setting up the AI for the enemies in the maze. | The integer values for difficulty will be determined by the program and will work on a basis where the variable is declared using values for different difficulties (int difficulty = 3;). |
| showWinnerImage - Boolean | A true or false variable which will determine when the winner screen is shown. |  |
| showEndImage- Boolean | A true or false variable which will determine when the end game screen is shown. |  |
| Minimise - JButton | A button used to minimise the window which contains the program. The use of buttons reduces the ability for incorrect data being entered and the program crashing. | The button works using swings JButton so on click a desired pre-determined process is carried out and in this case the window will minimise with no data input from the user. This means no validation is required. |
| exit - JButton | A button used to exit the window which contains the program. The use of buttons reduces the ability for incorrect data to be entered and the program crash. | The button works using swings JButton so on click a desired pre-determined process is carried out and, in this case, the window will exit with no data input from the user. This means no validation is required. |
| frame - JFrame | Used to create the frame of my program and is used to change the shape, size, and location. Also contains all panels, labels, and buttons within it. | The JFrame works using swing and will have set dimensions which cannot be changed by the user and therefore requires no validation. |
| itemScore - Int | A variable to hold item scores for objects. | The score for each variable will be generated between two numbers by the computer and will not require validation as it is not input by the user. |
| Menu - Class | The class which will welcome the user to the game and create objects with similar attributes and behaviours. |  |
| In Game UI - Class | A subclass of the menu class which will be used to represent the in-game screen and keep track of the objects within the maze. |  |
| Game Object - Class | A super class of Items and characters. Used to get locations of objects and retrieve their appearances. |  |
| Items - Class | A subclass of Game Object and is used to get the score of specific items and declare item scores. |  |
| Characters- Class | A subclass of Game Object and is used to declare and manipulate character and enemy directions. |  |
| Enemies- Class | A subclass of Characters and is used to move the enemy objects and check whether a specific enemy is on the character. |  |
| Main Character - Class | A subclass of Characters and is used to move the main character or detect if you have moved to an item. |  |

## Testing Data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | What am I testing? | How will I test it? | Expected outcome | Justification |
| Maze Generation | I am testing the level generator function within my game design to randomly create levels. I will test the consistency of the maze generation when running the game. | Normal Data- Start the maze generation from coordinates at the centre of the dedicated area for the maze. | The level should be randomly generated without any errors and should function as intended. | This test is required because the location of the beginning of the generation is chosen randomly. This means that it is required to check the maze can be consistently generated from the centre. |
|  |  | Boundary Data- Start the maze generation at coordinates at the edge of the area dedicated for the maze. | The level generation should take about the same amount of time to generate the level and there should be no errors. | This test is required because the location of the beginning of the generation is chosen randomly. This means that it is required to check the maze can be consistently generated from the edges. |
|  |  | Invalid Data- Start the maze generation at coordinates located outside of the dedicated level area which would not be in the array. | The level should not execute and will run into errors due to values not matching the values required to follow the generation process. | This test is required to make sure the game doesn’t attempt to continue executing incorrectly. The game will stop executing and fail however there will be bounds set to prevent this from happening. |
|  |  | Erroneous Data- Enter the coordinates in the form of a String. | The function should return error messages due to the function not being able to manipulate strings when creating the maze. | This test is required to make sure the game doesn’t attempt to continue executing incorrectly. The game will stop executing and fail however there will be bounds set to prevent this from happening. |
| Character Movement | I am testing the function which will assign keys to different movements the character can do. I am testing the inputs used to move the character. | Normal Data- Using the arrow keys as the input method. | The function should cause the character to move in the direction represented by the arrow on the button pressed. | This test is required to make sure the character can be moved using valid inputs from the keyboard. This is required for the character to navigate the maze. |
|  |  | Invalid Data- Using other keys instead of the arrow keys to attempt to navigate the maze. | The function shouldn’t receive the inputs from these kays as they have not been initiated in the function. | This test is required to check the game won’t receive invalid inputs and still operate as the same process being mapped to different keys may cause issues when running the game. |
|  |  | Boundary Data- Using two arrow keys at the same time. | The function should receive the input pressed first and then change to the input pressed second. | This test is required to make sure the game registers one input at a time and doesn’t change direction simultaneously as the shape of the maze will be straight edges and there is no requirement to go diagonally. |
| Score Counter | I am testing this function that will contain the score that the player has achieved during a single game. I will test how the function reacts when different numbers are added to the score. | Normal Data- Entering the number within the Long data range. | The function should add this number to the existing score and output it too the screen. | This test is required to make sure the game can take an input of a valid score and output it too the screen for the user to see. |
|  |  | Boundary data- Entering a number that is the largest the Long variable can possibly hold. | The function should output this number correctly and output a message to the user to say that the maximum score threshold has been reached. | This test is required to make sure the function can receive the largest number possible and respond appropriately. |
|  |  | Invalid Data- Adding a number to the score counter which would cause it to become too large for the variable to hold accurately. | The function should receive this value to be added to the score counter and identify that this will cause the score counter to exceed its highest value. The function should output telling the user they have exceeded the score limit and the maximum score has been reached. | This is a required test due to the maximum value for the long value being a specific number and each score that will be added to the counter being a rounded number such as 10, 20, or 100. This means that ultimately the final value will not be exactly met meaning a solution to an overflow of score is required. |
| Move Enemies | I am testing that the enemies in the level can navigate the maze easily and effectively. I will test that the enemies will only move to tiles within the level which are valid. | Normal Data- A valid location in the array will be passed to the enemies for their next move. | The function should receive the next valid location and move to this next location. | This is required to make sure that the enemies can navigate the maze when given valid coordinates. |
|  |  | Invalid Data- An invalid location in the array such as a wall will be passed to the enemies for their next move. | The function should receive this next location as an option and identify that the enemy cannot move to this location and then attempt another location. | This test is required to make sure that the function can identify when the next location is not valid and can identify a next move that is valid. |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Post Development Stage

|  |  |  |  |
| --- | --- | --- | --- |
| Main Function | What am I testing? | How I Will Test This | Justification |
| Maze Generation | I am testing that the level is generated at random and is different for each generation. | I will test this by entering the game multiple times and selecting different difficulties each time. This will test how my program/function will respond to these different scenarios and how this impacts the maze creation. | This test is required to check that the program can reliably generate random mazes/maps for the user to navigate. |
| Score Counter | I am testing that the game functions as intended when the game has to handle multiple collisions. | I will test the score counter function by navigating the maze and collecting multiple items. I will also test the impact of collecting an item and being caught by an enemy at the same time. | This will test that the score counter adds the scores reliably and correctly. It will also test that the game will not malfunction if multiple things impact the character at the same time. |
| Movement | I am testing the movement function by using the standard input which I have chosen (the arrow keys) and trying to navigate the maze. | I will use the arrow keys to manipulate the characters movement. | This will test that the characters and enemies will not violate the rules of the game such as going through walls and that the arrow keys can be used to manoeuvre the character. This will also allow me to see how the enemies respond to the movements. |
| Detection (Character) | I am testing that the program responds correctly when colliding with different objects | I will test this by moving the character object next to walls, on items and on enemies while in a game. | This will show me how the character object responds to these different objects and whether it will respond correctly. |
| Detection (Enemy) | I will test that the enemies can accurately locate and follow the main character. | I will test this by moving the character around the maze. | This will test that the enemies can respond correctly to the character’s movements and depending on the difficulty selected find a suitable path to intercept the character. |
| Instructions | I am testing that the instructions can be accessed easily. | I will test this by selecting the instructions button in different manners for example pressing it once, pressing it multiple times, and pressing it as fast as possible. | This will test the instructions will be displayed correctly under every circumstance consistently. |
| Difficulty | I am testing that when a difficulty is chosen it Is applied correctly to the game to change how the enemies move. | I will test this by selecting each difficulty and measuring the time it takes for the first enemy to reach the character on average. | This will test the functionality of the difficulty buttons to check they influence how the enemies track the main character. |

# Part C

I decided to break my development into eight main stages.

Stage 1- Creating A Window.  
Stage 2- Customising My Frame.  
Stage 3- Creating the In-Game Window.  
Stage 4- Creating the Level Generator.  
Stage 5- Creating the Main Character.  
Stage 6- Creating the Enemies.  
Stage 7- Creating the Items.  
Stage 8- Score Counter and End Game Screen

These stages represent each of the main functions within my game. The decision to divide my game into these stages was to make programming the solution more manageable and simpler. Each stage was decided upon by making it so each of the stages built on the last one so the development of my project will be easily measurable and there will be a very little requirement for backtracking as once a stage is completed it will not have to be altered again. This will always keep my project moving in a forward direction and reduce unnecessary distractions

## Stage 1

**Shape

Description automatically generatedGoals**

-Create a standard frame in which my program can operate.

-The window will be a specific size which will be determined when testing.

-The window will appear in the middle of the screen when the program is run.

When creating my frame, I tested the size that I set however I used the lengths of the sides the wrong way around, so the windows were too long. I corrected for this by switching the dimensions around as originally it was 500,1000 and is now 1000, 500

Text

Description automatically generated

Review – Overall this stage took little time and was quite simple. There was one trivial error that could be corrected rapidly after testing and the window carries out all the desired functions at this stage.

## Stage 2

**Goals**

The main aim of the second stage of development was to add some functionality to my program and window. This includes:

* The three difficulty buttons (Functionality may be limited in this stage)
* A title bar of a size that will be determined during testing.
* A welcome message
* Instructions that may be separate from the main menu if it is more appealing.
* Text

  Description automatically generatedDecide upon a suitable colour scheme to make the game appealing to the users.

Firstly, in stage two I declared a new JPanel and JLabel which will be used to represent the title section of my window. When testing I changed the background colour of the title bar to see what space was declared as the title Bar and I added the title image which conveniently placed itself in the center of the top of the window.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generatedI then decided to change the colour of the titleBar background to a colour which complemented the title more and I added a few more variables such as the JButtons and the JPanel (leftSide). As described by the name of the Panel it was created to contain all the items which would be represented on the left side of the window. I again changed the background colour of this panel to show which space was represented by the JPanel and I added the buttons.

However, after testing the new layout the buttons were placed next to each other instead of stacked on top of each other so I decided to test Gridlayout to see if this would resolve the issue.Text

Description automatically generated

After implementing the grid layout, I then tested the program again and got the result:Text

Description automatically generated

Although the grid Layout had resolved the issue with the buttons not stacking it had caused the buttons to seem compacted towards the left side. To attempt to resolve this I tried changing the dimensions of the buttons as the grid size depends on the size of the items placed in it.

Text

Description automatically generated

Text

Description automatically generatedI created a function due to the number of buttons this will be applied to. The function will increase the width of the buttons causing them to look less squashed.

I noticed that the buttons had some default features that were not aesthetically pleasing so I decided to change the font, colour, border and remove the small box that can be seen in the last screenshot which shows a small box around the word when the button is clicked. When attempting this a few errors were run into when trying to change the border.Text

Description automatically generated

Text

Description automatically generated

As shown here the createLineBorder was causing an error because of the absence of BorderFactory meaning it was undefined. This was solved with a bit of reading through the error messages and realizing the statement was incomplete. However, after this error was overcome the program executed correctly displaying:Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidenceAfter achieving to change the design of the buttons I decided to set my colour scheme to blue and black and add a welcome note when loading up the menu which would help the user find the instructions and give them a nice welcome to the game. Initially I attempted to add text to my newly created JPanel using a JLabel however this would only work with short messages as it trailed out of the window.

I then tried using a newly discovered method of using JTextArea instead of a JLabel.Text

Description automatically generated

The JTextArea allowed for the use of \n to create new lines whereas JLabel did not which allowed me to fit all the text in the desired area.

A screenshot of a computer

Description automatically generated with medium confidence

(Add instructions button documentation from home)

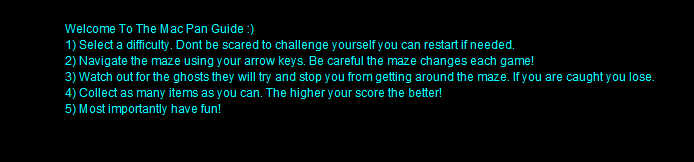
Once I had added the new text box I tested my program and quickly realized I had not set my new guide textbox to be not edited when run.

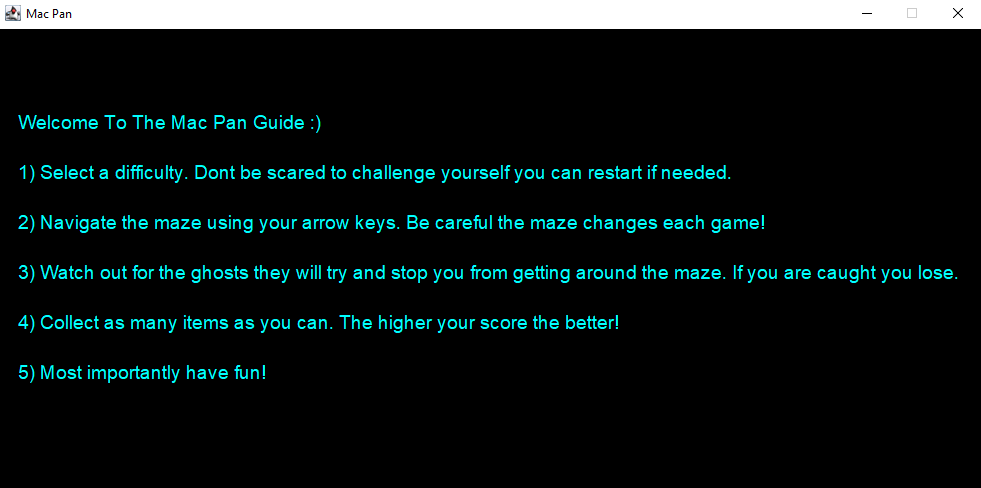


I fixed this with one simple line of code which causes the text area to not be editable by the user when the program is running.

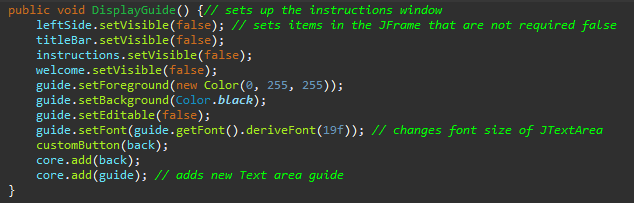
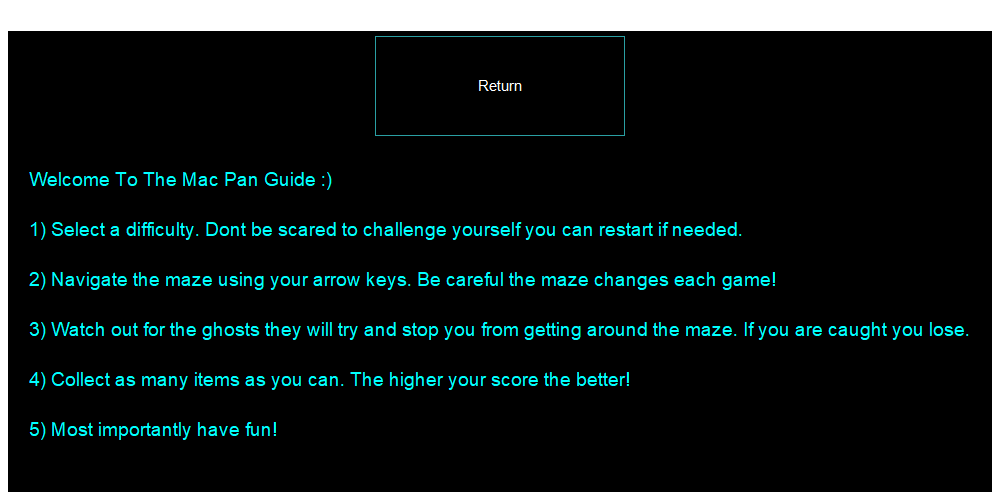


After setting the text area to not editable I filled out the instructions and changed the colour to match the title screen. This allowed me to keep the desired colour format. I achieved this by typing the text into the JTextArea defined earlier (guide).



The correct text was output in a nice listed format however the text didn’t fill out the screen and seemed quite cramped into the centre. I then tested different font sizes before settling with 19 as it managed to fit nicely in the frame.

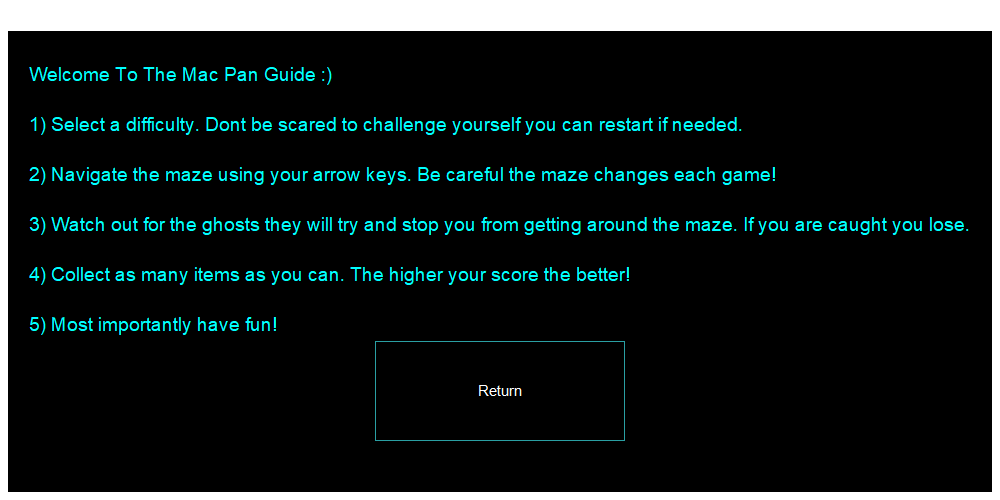
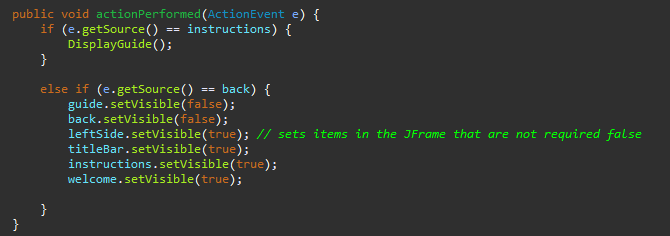
I next added a return button for the user when they have finished reading the guide and want to pick a difficulty. The return button (back) would follow the same layout and appearance as the previous buttons and would appear at the bottom of the guide frame.



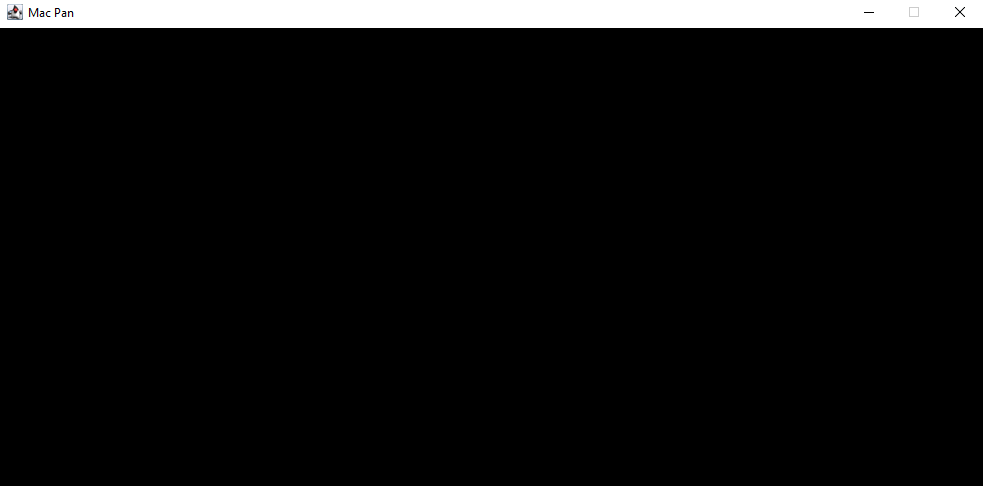
After testing and running the program the return button appeared above the guide text meaning that in my code the button had been added to the core before the guide JTextArea meaning I would have to change the two around.

This was easily corrected by simply adding the JButton after the JTextArea: 

This resulted in the JButton appearing below the JTextArea as desired. I also added some functionality to the button by causing a click of the button to return you to the menu screen.



I decided not to create a separate function to revert the screen back to the menu frame as it was only specific to reverting the guide screen back so this way the function would never be repeated and would be pointless. I tested the functionality of the button by pressing instructions, return and then instructions again. On the first instructions and return press it worked successfully however on the next instructions button press this was displayed:

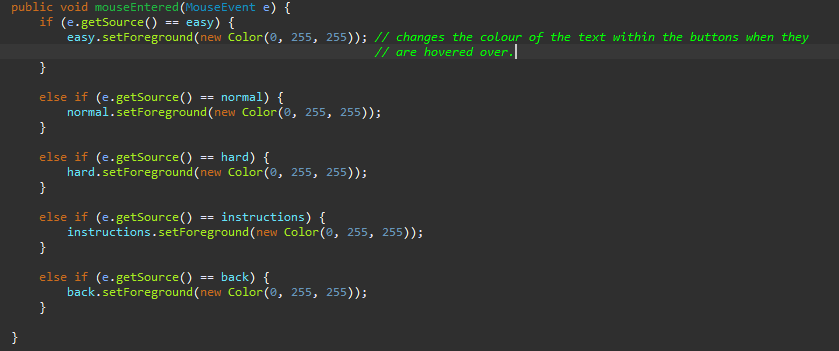


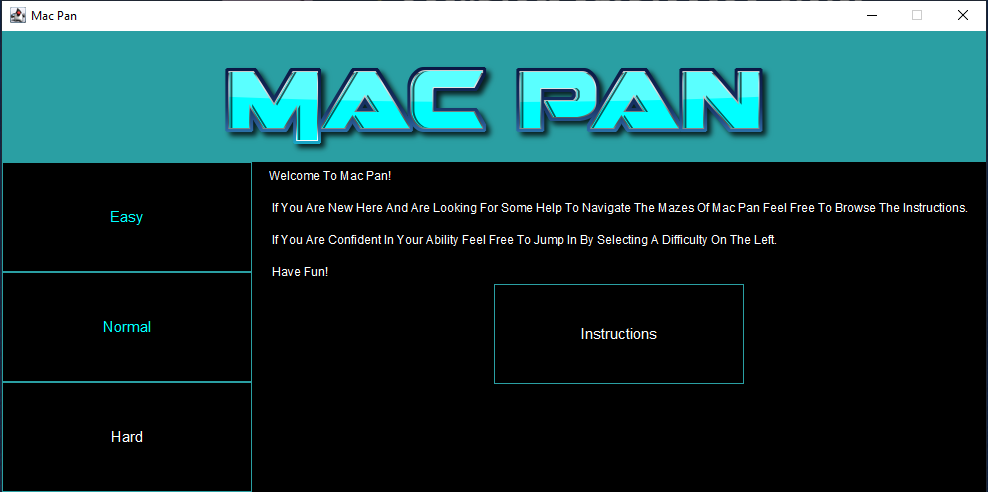
As you can see there was an issue redisplaying the guide and return button. I overlooked my code that processed how the instructions were displayed which brought to my attention that guide and back are not declared as visible as this is automatically done when adding them to the JPanel the first time.

After the back and guide were set as visible this resolved the issue and the menu and instructions panel could be moved between as many times as desired without fail.

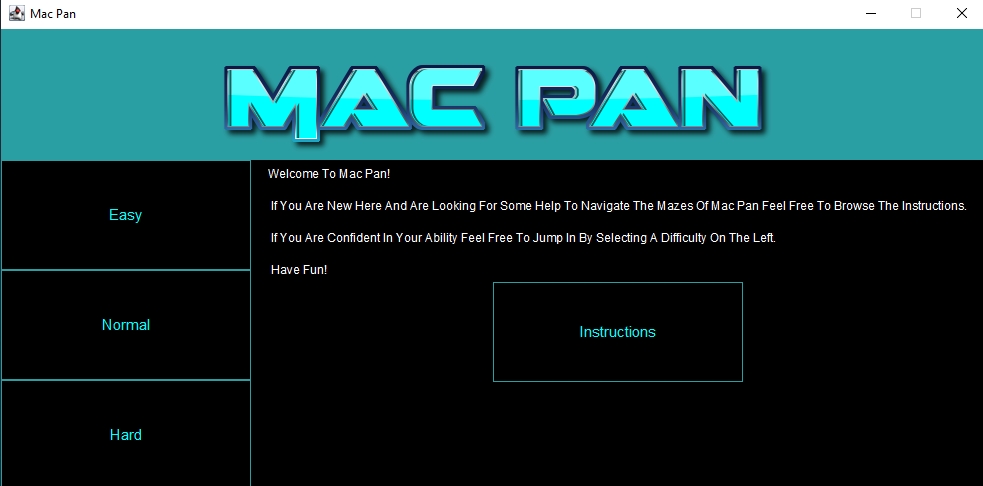
After ultimately finishing the layout of my menu I realised it wasn’t necessarily obvious a button could be pressed when hovering over it so I decided to make the text colour change when hovering over the button.

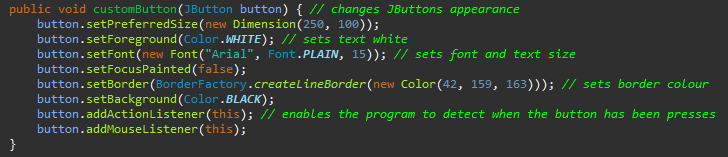
I initially tested that when the mouse enters the area of the button it would change colour. I knew that when exiting the proximity of the button it would remain the same colour as I hadn’t altered the mouseExited function. However, this was just a quick test to check that what I had programmed was working as intended so far.





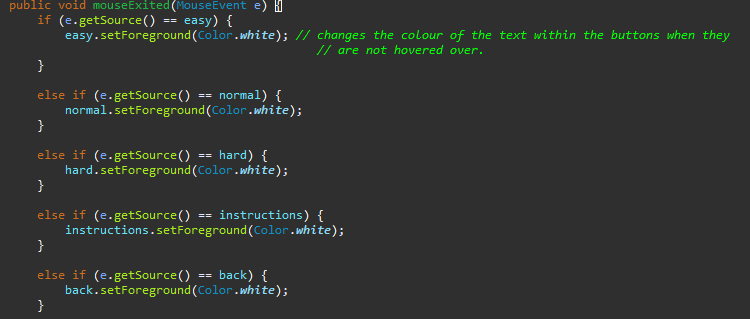
This is the output of my test where I hovered over two of the buttons to show the difference between a button that has been hovered over and a button that has not. I did test that every button changed colour.





I also decided that because each button would have an action listener and mouse listener I would add the lines of code which added them to each button to the function customButton. This would reduce the amount of lines of code in my program.

I next added to the mouseExited function to make sure the button text changed back to which once it wasn’t being hovered over anymore.



**Review**

During the development of stage two I achieved to reach all of my goals set at the beginning and add some functionality to my window. There were a few major errors along the way which stopped my code from running such as the absence of BorderFactory however it was nothing that couldn’t be overcome. Towards the end of the development of my menu I realised that it would improve usability to make the buttons change colour which wasn’t mentioned in my goals but was quite important. I also decided as mentioned in the goals that I may not add functionality to the difficulty buttons in this stage as it was more suited to the next stage.

## Stage 3

Goals

The goal of this stage is to create a larger square window in which the game will be played. Although not added in this stage the window will contain the maze, score, enemies, main character and all the other aspects of the game.

Firstly, I created a new class named GameUI. This class control everything from the appearance of the game to all the item’s locations within the game. I then added functionality to my difficulty buttons by making it so that if you pressed one it would call the display function in the new class.

Text

Description automatically generatedThis code was added in the actionPerformed function. I have declared and assigned a unique value to difficulty depending on the difficulty chosen which will be referred to later by my program to set the enemies ai.

Due to GameUI still requiring variables and data from Menu it extends the menu class.Text

Description automatically generated  
However when trying to run my program after declaring my new object for my new class within the class it is extending it caused a stack overflow error because the class that extends the other when declaring a new object of the new class inside it, set an infinite recursive loop.

Text

Description automatically generated

I fixed this by creating my GameUI object in a function, so it is created only when the function is called.

Text

Description automatically generated

This would call DisplayGameFrame(); and create a frame which would appear in the middle of the screen and not be resizable. I also added the core panel so I can add items to my frame later.

Text

Description automatically generated

Graphical user interface, application, website

Description automatically generated

Although the new frame appeared as desired and in the correct location the menu remained visible which had to be changed.

Setting the frame to not visible once a difficulty button had been pressed solved this issue leaving the output as just the new Game frame.

Graphical user interface, application, Word

Description automatically generated

I next decided to test my program by painting the core panels background. The purpose of this was to check that the panel was added to the frame and to show what space it occupied.

Shape

Description automatically generatedAs shown the core panel occupies the whole frame as a position for the panel was not specified. This is fine for now.

Review

Stage 3 came along relatively smoothly as it was only a small stage to make sure the next frame is ready for the game. As shown, there was an issue with a stack overflow due to an infinite recursion which caused a slow in progression as it wasn’t as obvious to fix as some of my other mistakes until I did some research about the cause of stack overflows. This became a relatively easy fix once the issue was made obvious that I was exhausting the memory storage assigned to my program. Once the fix was applied it became quite easy to set the location and background colour of my frame /panel. Overall stage 3 was quite short and quick to complete due to its lack of content which will be added in later stages. The main purpose of stage 3 was achieved which was to create a sturdy base to build on in later stages.

## Stage 4

Goal

-Create the maze class  
-Create a function within the GameUI class to access the maze generator  
-Create a function in the maze class to create the array for the maze  
-Create a function that can be called to create a maze block object   
-Create a function that can be called to create a pathway block.  
-Draw the maze in the GameUI frame

I started by creating my new class which was named MapGen. This class conatained a function createMap which I for now added a print statement to print out “Made it”. I did this to test whether the new function I created in MapGen was successfully called from GameUI.

Text

Description automatically generated

Text

Description automatically generatedGraphical user interface, application

Description automatically generated

The output was successful meaning the program was working correctly so far. I had decided before not to make MapGen a subclass to GameUI due to the fact there was no requirement for any of the variables in GameUI in the new class.

I next replaced the map object with two new ones as I realized it wasn’t required. These two new objects represent a wall tile and a pathway tile which have their values declared into variables in the MapGen class. I have also declared a 2d array which is of size 100, 100 and can be changed if required in the future.

Text

Description automatically generated

Text

Description automatically generated

After declaring the 2d array I created a nested for loop to make eack index in the array conatain a value of 0. This value of 0 would represent a wall tile and once the map creating algorithm completes (which hasn’t been made yet) a 1 will represent a pathway. I also printed out the array to test that every slot contained a 0.

Text

Description automatically generated

Background pattern

Description automatically generated

The output of the test was 10000 0’s which was what I was aiming for (in a square shape of 100 by 100.)

Following the test on the array I declared some new variables which would be used to generate and store two random numbers between 100. These random numbers would be used as a starting point for the maze generation to begin. Similarly, to the array test I also output the two random numbers generated to make sure the y remained in the bounds and would be different numbers as this would increase the randomness of the map.

Text

Description automatically generatedOutput:



The two numbers output were different and within the bound which was what was expected.

Text

Description automatically generatedThe next part of this stage was to start generating the maze within my array using 0’s and 1’s. I started out by declaring some new variables:

These included lines 12, 13, 14 and 15. The decision to use ArrayLists was since because my maze is generated randomly it is unknown how long these arrays will have to be.

I then created a process that would carry out the maze generation for 4 cycles to check that it was running fine.

Text

Description automatically generated

Text

Description automatically generated

As shown in the screen shots I added in some print operations so when I ran my program for a test, I could see the inner workings and operations of the program as it executed.

The test:

In the screenshot of the test output there is a list of numbers, the left is the x location in the array and the right is the y location in the array. The first pair is the start location, and the rest are potential locations which have been selected as the next location. This is where I have run into my first issue that as you can see some potential locations are being repeated which is not supposed to happen and causing my array to be output incorrectly.

A screenshot of a computer

Description automatically generated with low confidenceThe next test I carried out was to print out the maze and check the array to see if a maze had been created. For this I reduced the mapArray down to a 10 by 10 so it was easier and quicker to determine if a maze had been generated correctly. I also started with 10 cycles of the for loop to see if there were any obvious errors.

As shown here the 1s are a path and 0s a wall. At only 10 loops through the process there should still be a single pathway however as you can see there was 1’s that are surrounded by 0’s. This was not supposed to happen, so I added more print statements to attempt to follow the process as it executed.

A screenshot of a computer

Description automatically generated with medium confidenceThe outputs corresponding to the output above are.

As you can see it specifies the first node, the next node and the other potential nodes again and again until it is complete. When following these numbers through I realized they were all correct and the finding nodes side of the process was working as intended.

Text

Description automatically generatedSo this caused me to look into how the current and new nodes were being manipulated and changed throughout the process which brought my attention to this block of code.

The block of code above works to join the new node and current node once the new node has been identified. However, when I coded and calculated the math’s required to do this, I made a fatal assumption that the new node would be one produced from the current node which with the method I am using is not true the new node can be any of the potential nodes in the array. These broken math’s if statements were causing 1’s to be placed in completely incorrect locations.

A picture containing text, electronics

Description automatically generatedI firstly added two more array lists to store the column and row of the origin of the potential nodes and I added in two more print loops to display it was working correctly.

As you can see potential nodes 4,9 and 8,9 have a base node 6,9 which when more passes have occurred will node be the same for all potential nodes.

After correcting the math’s and declaring some new variables I managed to achieve an output of an array in which all the paths connected.

Text

Description automatically generated

Text

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

This was a good step in the right direction however there was still an issue of having potential nodes being repeated from different base nodes which basically means if I let my function to draw a maze run all the way through it would just create a grid of ones every time. As you can see in the array output there are already some squares of 1’s being generated.

I attempted to fix this issue by using a for loop to identify any repeated potential nodes on each run through which should have worked effectively however it caused an error message to occur.

Graphical user interface, text, application, chat or text message

Description automatically generated

Text

Description automatically generated

It was interesting that the programmed functioned initially up until a certain point in the loop. I tried removing the for loop and the programmed functioned as it did before which led me to believe the issue was that I was removing random data from the arrays by accident as I didn’t remove the old removing sequence once the loop had finished with the new node.Text

Description automatically generated

The code above was the code I deleted as it was incorrectly removing data from the arrays due to the new for loop.

A screenshot of a phone

Description automatically generated with low confidence The new outputted array was what was desired in which all the ones are connected to each other and there are no repeated routes (grid squares being produced). Now the next step was to scale this up and set it, so the loop/ function ends once the whole map has been generated.

Text

Description automatically generatedScaling up was easy all I had to do was change the size of the array and I made the loop end after one of the Array lists was empty as they all contained the same number of values at each point. I decided to go for a 50 by 50 array size as my frame was 1000 by 1000 so there will be enough room for the other features to be added after and so the map can be a decent size. I also made it so there is always 0’s along the edges by changing the values to 49 in the image below.  
The if statement to break the loop was placed at the end :Text

Description automatically generated

The completion of this substage also allowed me to remove the print statements used to test the code however I kept the output of the whole array at the end as this could help later.

Final output:

A picture containing background pattern

Description automatically generated

The next stage was to draw the maze which included creating a paintComponent method and using Graphics g. I have also decided to move the contents of the method previously called displayGameFrame into the Menu class as this meant I could manipulate the current frame instead of having to create a new one and it made implementing the graphics easier.

Text

Description automatically generatedText

Description automatically generated  
The class GameUI contains:

I set the color of g2d to red and created a rectangle to fill the frame to check it would work. Which it did.

After considering how I was going to draw my maze I realized that the pathBlocks object was not required as the character would be able to move with the absence of a wall tile and is not dependent on the availability of a path object.

Shape, rectangle

Description automatically generated

I next altered the createMap function to always produce a permanent border around the edges which would be represented with a number 3.

A picture containing background pattern

Description automatically generated

With this change my map was ready to be drawn using the function drawMaze in the GameUI class:

Text

Description automatically generated

With the creation of these for loops I realized I needed access to the dimensions set for my wallBlocks object. The object was changed to only require one value which was its dimensions as after the review the others seemed unnecessary.



Text

Description automatically generated

I also made some changes to the paintCompponent method by making it paint the background colour black and when all combined gave the desired output:

A picture containing text

Description automatically generatedAlthough now the map appears in the top left of the frame and doesn’t necessarily have a nice appearance the main goal of this test was to check that it was drawn correctly which was achieved.

Now that I was aware that the maze was being generated and displayed correctly i decided to try and make the maze appear nicer by making each maze wall tile an image.



Graphical user interface, application

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

I added the line of code to draw the image but due to me forgetting I set the block to certain sizes the output wasn’t what was desired.

Text

Description automatically generated

Next, I opted for a nicer image I found and had to implement a line of code which would change the size of the image to the size of 15 pixels by 15 pixels.



While attempting to change images for the walls and implementing them into my maze I ran into a what seemed rare error where some of the threes in my array were being replaced with other numbers resulting in an output which didn’t have a left barrier.

Qr code

Description automatically generated

To fix this I simply added another stage of validation to check that the new pathways or walls were not replacing the pre-generated barrier. It simply checks that the next spot is not a 3.Text

Description automatically generated

Next was too just place the maze in the desired location on the screen. All I did for this was find the midpoint and move the maze by changing the x and y values to 117 as shown.

Text

Description automatically generated

Qr code

Description automatically generated

Review

Stage 3 was the most challenging stage so far due to the math’s behind generating a random map map within a set area. There were a couple of issues with my program not executing due to infinite recursion and just pure math’s errors which caused it to be inconsistent in generation. The recursion issues were relatively simple to resolve due to the program telling me the memory was being used up however the math’s error took a little longer to resolve and required the use of a few trace tables to identify the part of the program which was executing incorrectly.

Overall, the implementation of the MapGen class was smooth and the use of 2d graphics wasn’t too demanding.

## Stage 5

Goals

These include

-Adding a character  
-Adding a way to manoeuvre the character  
-Making collision detection for if the character is touching a wall  
-create an appearance for the character which presents the appearance of an animation.

First, I created to new class GameObjects which would contain a few functions and variables which would be a superclass for the character class.

Text

Description automatically generated

Next, I created the characters class which extended the GameObjects class and added some starting variables and a constructer. This allowed me to create my first character’s object main and draw a simple square to represent it.

Text

Description automatically generated

Text

Description automatically generated

A picture containing qr code

Description automatically generated

After creating the square to represent my character I started creating some functions to help with the movement of my character. I first added some more variables to help with keeping track of what direction the character is moving.

Text

Description automatically generated

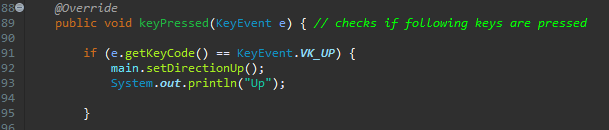
I then implemented KeyListener in the GameUI class which allowed my program to listen out for key presses which gave me the use of methods such as keyPressed.



Text

Description automatically generated

I then added a print statement to check whether me pressing the button was being registered by my program.



However, when pressed not output was yielded and caused me to decide to change to the use of Key Binds instead of action listener as it was better adapted for swing. This caused me to delete all my previous KeyListener functions and create a new function and add two newly declared items.

The decision to use WHEN\_IN\_FOCUSED\_WINDOW was since I didn’t want the game picking up any uses of these buttons in other programs.Text

Description automatically generated

This method was created to initialize the binds and contain their action if they are pressed as of current it only contains the downward movement to check if it worked. The method is called from the Menu class.

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing text, electronics, file

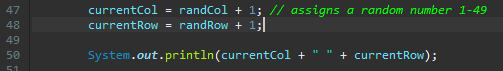
Description automatically generated This test shows the block placement in the window and the coordinates for its position next to it. As shown in the screen shots the block can now move downwards and the other directions must be added.

Text

Description automatically generated

After checking the downwards action worked all that was required was a repetition of the functions with a few adjustments such as setting x for left and right instead of y and their names. When testing they all functioned as intended a minor error was found in which when right was pressed the object moved left. This was due to setX(); containing the incorrect sign in front of the five.

When testing the movement of my character I found that my maze generated incorrectly when the starting location was selected as a wall tile. This caused the maze generation to end early and most of the maze was left full wall tiles.

I identified the cause of the error by adding a print statement that output the initial location chosen for the maze generation to begin at.  


This yielded the output:



This shows me the column and row that has been chosen and helped me identify that the issue occurred when a border location was selected.

I fixed this by making sure that the random number could only select a number from 1-49 so it does not appear on the boarder.

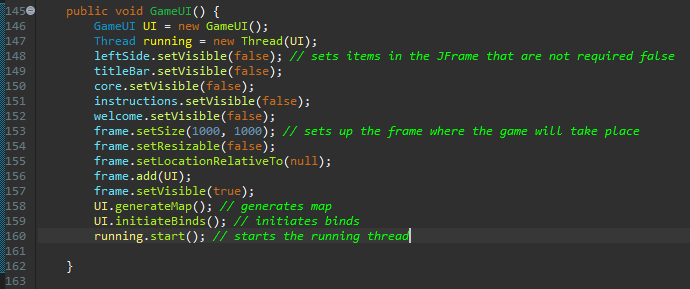


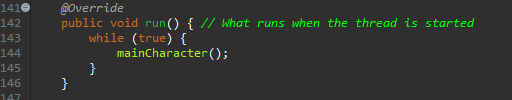


I wanted the main character to move continuosly until his direction is changed meaning I needed a thread to monitor his movement as the game progressed. The idea of the thread is to carry out all of the changing parts of the game as it is played. I had to firstly implement runnable into my GameUI class to enable me to use threads and the methods that come with them.

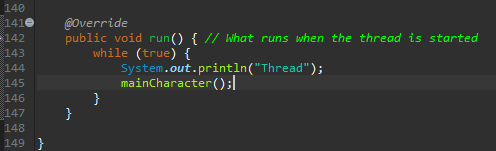


I first changed my running method into the run method which is executed when the thread is started. I initiated the thread variable running in the menu class and added the statement ot start the thread.





This was supposed to make the character move at a constant rate until another direction was chosen however this had no effect on the character and it remained stationary. I attempted to add print statements to identify what parts of the program are actually being executed and this made the thread work as intended.



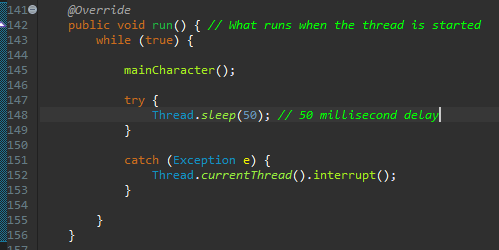
Once adding the print statement, the main character was able to move at a constant rate once the arrow keys had been pressed. I found that this was to do with synchronisation and happened because the thread assumed that the variables it was operating on were not to be changed. However, adding the print statement caused the variables to become updated.

In the test below, I have pressed the up arrow and although it seems he has disappeared it Is because a timer has not been added so it moves too quickly.

As shown the main character is not present in his starting location after the press of the up button. Although this could be down to a graphics error I can verify his location using the coordinates output when he changes direction.

Due to the variables not being updated and adding a print statement is not a very effective method to update them I made the variables controlling his direction volatile. This meant they would make the thread aware of any changes to these variables.

I then added a delay so the position of the main character doesn’t change so fast and can be seen on screen by the user.



When run the character was only redrawn when an arrow key was pressed meaning I had to move the call to repaint from when a button is pressed to when the location is changed.



Once I had moved the repaint calls to the mainCharacter() method I found further issues such as the character moving up when down was pressed and so on. This required me to go back to my mainCharacter() method and button Initialisation method a change around some variables which had been named incorrectly.

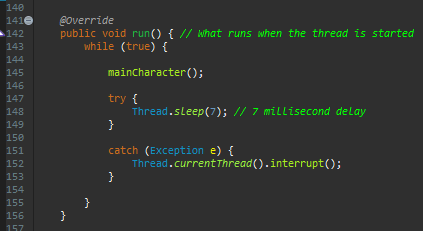
In initiateBinds() all initiated buttons set the direction to down so I had to change their variable names.

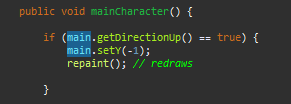


I also had to alter some values passed in the maincharacter() method.



After these corrections the main character was able to move in a given direction at a constant rate until the direction was changed. The only issue now was that it didn’t appear as a smooth animation due to the distance he was moving each time the main character method was run. I decided to change these values to find a suitable combination which created the appearance of a smooth animation.

I altered the values given in the thread sleep timing to 7 ms and changed the number of pixels moved to one each loop creating a smoother animation and the main character now moves at a reasonable speed.



Next, I intended to make the main character spawn within the map which would require me to change the initial location of where it is first drawn.

My pan was to use my map array to randomly locate a position which was not a wall block and draw the character there. This required some extra functions and variables to be created in the GameUI class.



Text

Description automatically generated

Then function initialCharacterPosition() relied on two more functions created in the Class MapGen.

A screenshot of a computer

Description automatically generated with medium confidence

When testing this locating function I had created, I found that the majority of times the game was started, the main character would be drawn in a valid location; however, every so often, it would be drawn on a wall tile as shown.

A picture containing text, crossword puzzle

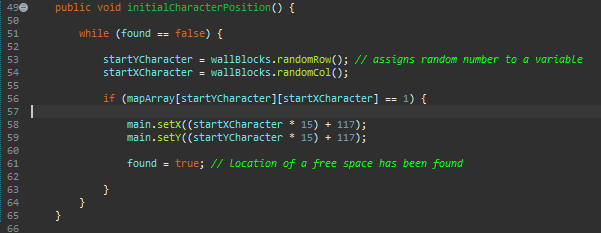
Description automatically generated

I decided to add some print statements to see what location in the array was being considered the character’s location. These yielded a list of numbers that described the column and row. The numbers repeated twice were the coordinates selected as the final location.

Text

Description automatically generated

Using these outputted values, I found that the rows and columns were getting confused. I had misnamed the variables, so all of the logic was corrected, but the values were being assigned to the wrong variables at the end.

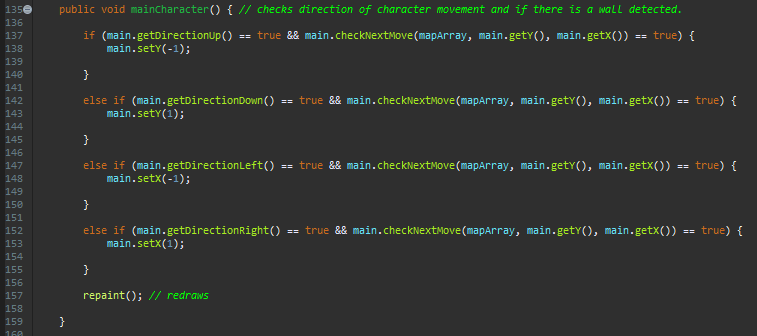


The above function which has had the variable names corrected and yielded a desired output each time the program was run where the character started inside the maze on the pathway.

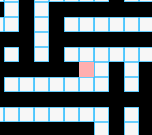
I was next required to implement some collision detection for when the character touches the wall. I created a new functions checkNextMove() which needed me to declare two new variables row and col. This function was created to detect that the next move would be valid.



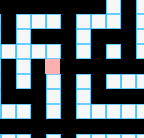
I then called this function in the mainCharacter() function.

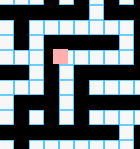


This yielded a varied output. When pressing right and up the program functions as intended and stopped at the walls.



However, when pressing down and left the main character stopped one tile too soon.

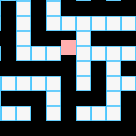


I realised that the reason for this behaviour was a combination of the integer always rounding down in my function and the fact that the location was being measured from the top left of the character. I tested my conclusions by trying to move the character up when the whole character wasn’t on the pathway. This test confirmed that the issue was what I stated earlier.

Due to how my game was generated to fix the issue that was when the character moved mostly into a new tile area it would still use the previous array data until the top left pixel enters the new area. I fixed this by adding some extra and statements into the if and else ifs. This added an extra requirement that had to be met before the function could decide a movement was valid.

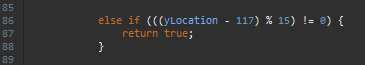
The extra && requirements that had to be met made sure that to make a movement the top left pixel was in the right position for a move to be valid.



Although this fixed the issue of the block going partially through walls there was an issue where if the user was to change their mind and for example are next to a wall and choose to go down and up fast the function would determine that the character cannot travel back up. This was due to the function not updating the array location as the top left corner of the character had not reached to top left corner of the next tile.

As stated as the example I have pressed down and up quickly where the character was unable to leave its current tile. The function then determines that the character cannot move back up as there is a wall above the current tile.

To fix this I had to add in another else if option that doesn’t require the location of the top left of the character to match the top left location of the tile.

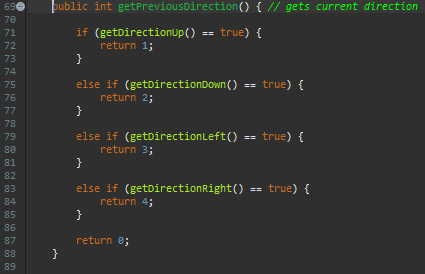


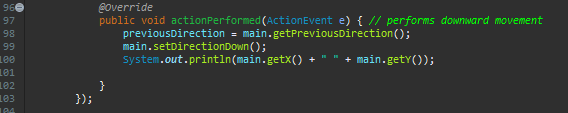


These are the two different types of else if statements I added in where the one that contains the yLocation was used if the character was travelling up or down and the xLocation one for if the character was moving left or right. This fixed the issue allowing the character to move freely.

When testing the characters movement previously I noticed that is was very hard to move down pathways which were in between two wall tiles. This was because the users timing when changing direction had to be perfect.

I wanted to implement a variable and method which would consider a direction change before the intersection is reached so the users timing doesn’t have to be exact. I had to start by making my character travel constantly if a valid path was not chosen because currently if I was to press left into a wall the character would just stop meaning a direction change before an intersection would not be valid. I started by creating a new variable in the GameUI class and adding a new function to the Character class which could be called when the direction is changed.

The method called compares the current direction and returns a value corresponding to the direction. This becomes the previous direction as this method is called before a direction change is made.



Each of my actionPerformed methods for each direction was given the line of code to execute before the direction change to store the last direction.

The next step was to add an else if statement that would consider the previous direction is the new direction was not valid.



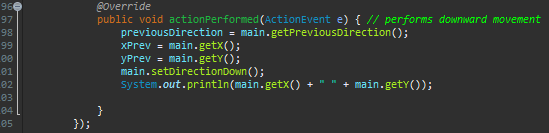
I added print statements to check the logic of the program when it was executing. When testing I found that if the character reached a dead end and the previous direction was the opposite way he would bounce of the wall.

For example, here the character started in the middle of this path. I pressed down and the up and the character travelled to the top where he reached a dead end. The program then considered the previous direction which was down and caused him to finish at the bottom without and extra inputs.

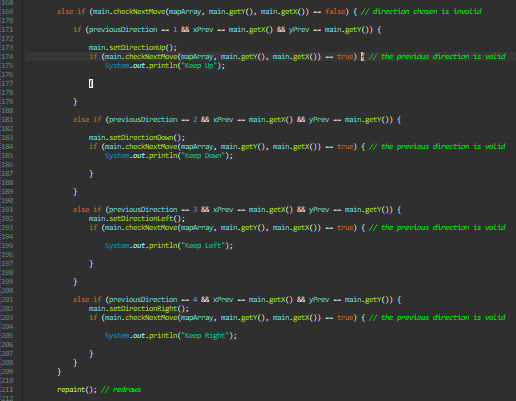
Console Output:

Due to this issue I implemented two new variables in the GameUI class to be considered when considering the previous direction.



yPrev and xPrev will be called when getting the previous direction so when a dead end is met these variables can be considered.

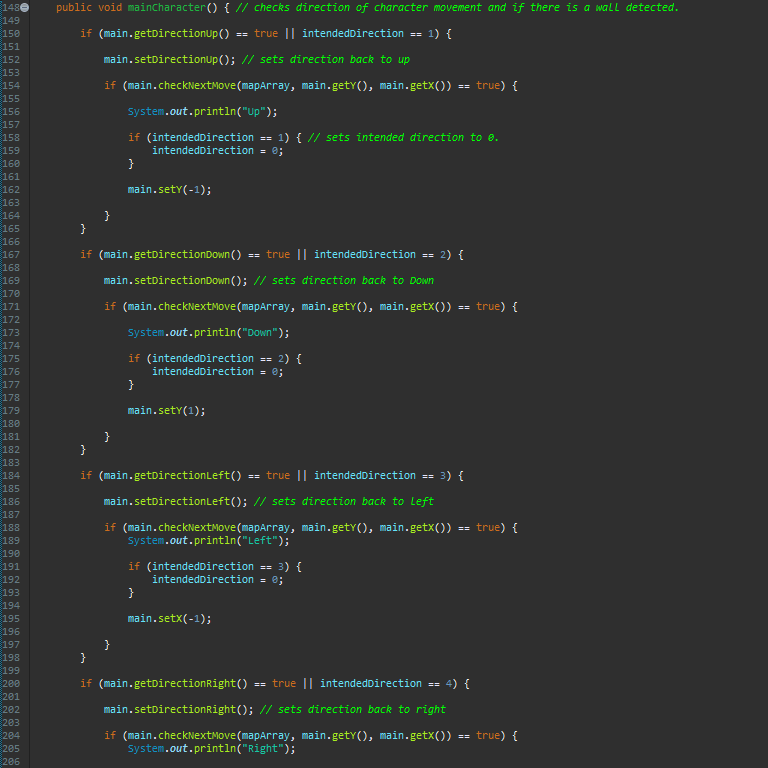
Doing this allowed me to add the following to my if and else if statements.

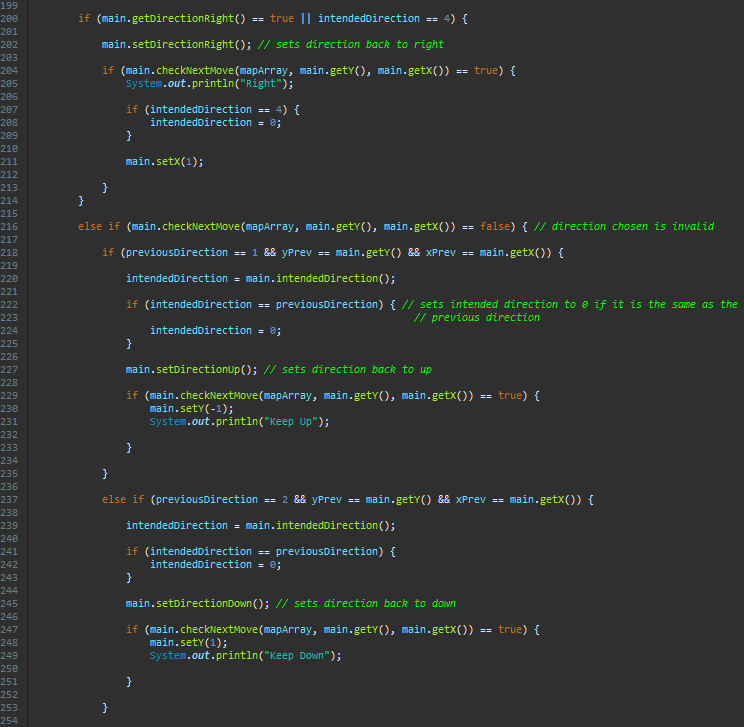


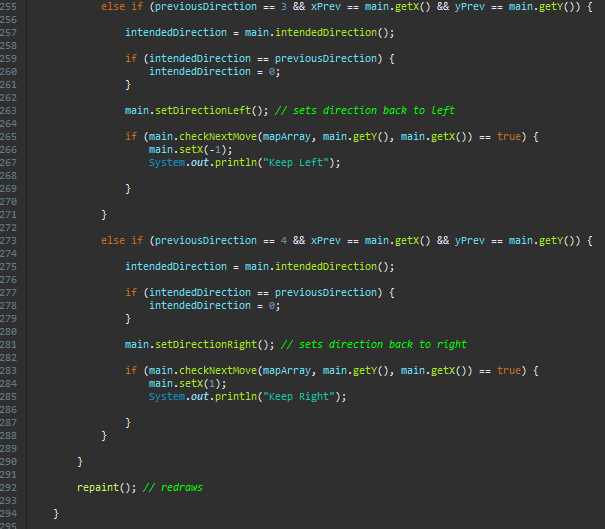
Adding these in meant that a previous direction would only be considered if the invalid direction occurs at the same coordinates as the previous direction is assigned. This fixed the issue of the character travelling the wrong way when a dead end was met.

Next, I am going to implement the idea mentioned earlier which means the users timing doesn’t have to be perfect when turning a corner.

I created a new variable intendedDirection which would store the direction the user wants to move next and set it to 0 along with a new function in the characters class.

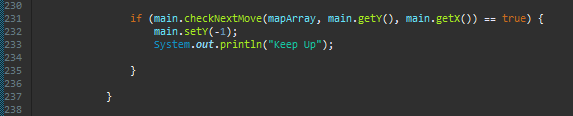






When adding in the intended direction I had to change some of my else if statements to if statements. This was because I have made use of OR in my if statements which means one statement may be seen as valid and then part way through the statements execution it is realised another one must be tried. The OR statements were put in place to check if the character can move in the direction wanted by the user.

Part way through each statement I had to set the direction to make sure the logic of the operations was kept in check. For example, if an intended direction was being checked in the if statement it would then check if the move was valid based of a different variable which would yield incorrect data. I also added in if statements to check what the intended direction was and whether it should be set to 0. This was due to the logic of the program dictating that if a dead end was met and the intended direction was the same as the direction of travel if would attempt to keep the character moving in this direction bringing the character to a standstill. There is also an if statement to set the intended direction to 0 if the character has moved in the intended direction which meant that if another intended direction was not chosen the character would not remain attempting to move the old intended direction.

However, when I tested the character’s movement after the addition of the intended direction I found that if I pressed an invalid direction the character would no longer move in the previous direction and would just stop. While reviewing this issue I found that some of the code within this section was actually redundant and had no effect on the outcome of the method. The else ifs that were checked if the next location was false had an if statement in each one that served no purpose.

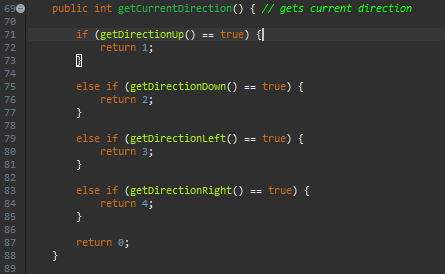
This if statement changed no direction variables and the main.setY() would be carried out regardless of if it existed within the if statement.

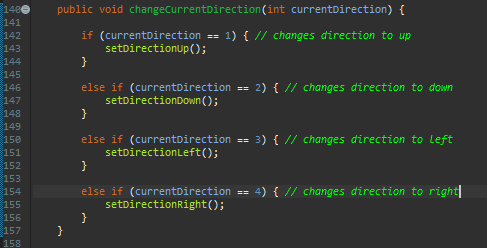
Due to the previous direction variable only being change by button presses this caused an issue as the program changes the direction of the character to check possible movement options. The previous direction variable can therefore only be used to correct the direction for the first cycle after the button press but after that the variable becomes inaccurate.

This issue caused me to consider adding in another variable which would keep track of the current direction of the character regardless of a button press. The idea is for this variable to maintain the correct direction when the previous direction cannot and the intended direction remains the same.

I created a new variable called currentDirection and altered an existing function originally named getPreviousDirection. Due to this function doing exactly the same thing as the new function I wanted to create I renamed this function getCurrentDirection. I also created another function called changeCurrentDirection.

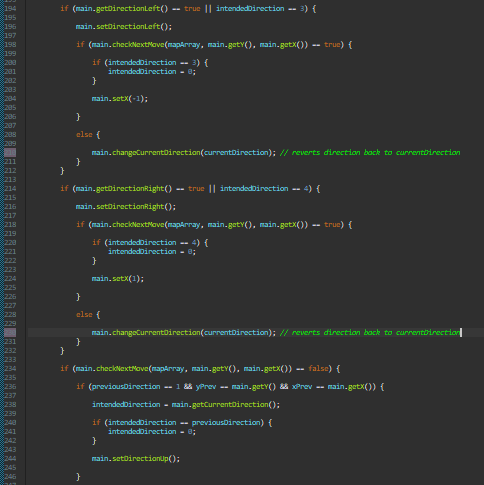




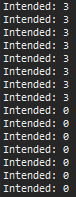


I added an else statement to each if statement that considered the intended direction. This else statement is used to revert the direction back to the current direction if the intended direction is not valid.





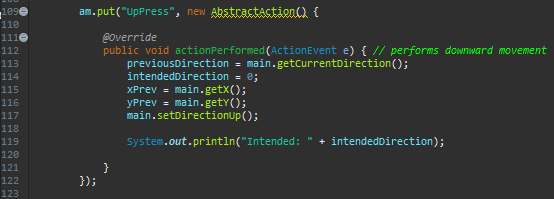
When testing the current processes worked as intended and as shown below the intended direction stored the direction the user wanted to travel until it is possible.

 As shown here the intended variable stores the wanted direction which in this case is right until it is a valid move. The program variable is correctly and accurately stored through each cycle until it is met.

Although while carrying out this test I found that if I pressed a direction for example left the intended direction was made left. This works as intended but if I was to change my mind before the left move was valid and say I wanted to continue travelling in the direction I was going. If I was going up and I pressed up to continue travelling this way the intended direction would remain left even though this was no longer my intended direction.

My first attempt at fixing this issue was to convert the variable intendedDirection to 0 on each button press. Although this becomes more work for the program as the intendedDirection variable has to be reassigned each loop after a key is pressed. This should fix the issue.

I tested my new slolution by adding print statements to output the intendedDirection each time it changes.



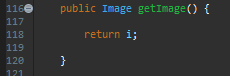
An example of the downward method is above which shows what each direction method contains excluding the obvious fact that in left it would be main.setDirectionLeft(); for example.

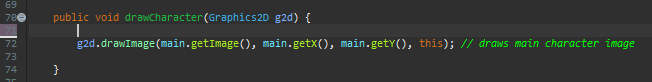
As shown here I have chosen a valid direction to move so the intended direction is 0. I then chose to move right setting the intended direction to 4 but then chose to carry on travelling in the original direction which sets the intendedDirection back to 0 as it is a valid direction.

This solution worked as intended allowing the user to change their mind about what direction will be travelled next.

My next step was to create a more interesting character as currently he is only represented by a pink block. I chose to create a simple circle shape with a grey outline and blue inside. I had to create two variables to store the image and resize it which would be accessed by the main character object by calling the newly created function getImage(). These new variables and functions were declared in the character class. I also altered the drawCharacter() method in GameUI class so the new image is drawn instead of the pink square.









When testing to see the appearance of the new mac pan character I realised it was quite hard to see it due to the colour schemes being so similar. I decided to change the colour of the wall tiles and surrounding tiles to make the main character stand out more.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Qr code

Description automatically generated

While checking the appearance of the main character I also decided that he was clear enough and didn’t stand out as the main character. I redesigned the main characters as more of a square shape and used some bolder colours.

A picture containing building

Description automatically generated

Review

Stage 5 included most of the main development for my main character. I first stated my goals at the start of this section in which I met 3 out of 4 of them. I successfully implemented a character that could move through the maze and detect collisions with the walls. These sections were very heavily based on maths and long calculations. Most errors generated weren’t necessarily that the code couldn’t run but it would run incorrectly meaning they were mostly logic errors. This stage also included an extra step where I redesigned the wall and edge tiles for the maze due to viewing issues. The one-stage not met was adding animation to my character. This was not necessarily challenging but instead pointless as the main character is 15 by 15 pixels which is too small for a face of some sort to be on the character. Creating a larger character to make animation more realistic would compromise the complexity of the level which I do not consider worthy just for animation. Due to this, I would not be able to make a meaningful animation for the character.

## Stage 6

Goals  
-Create 4 enemies (May change depending on testing)  
-Make the enemies move throughout the maze  
-Make the enemies try and catch the main character  
-Add some more complex decision making to make sure catching the main character isn’t too easy.

Firstly, I was set on creating an enemy and making them appear within the maze. I created a new Characters object named enemyOne and proceeded to draw the object onto the frame.



Text

Description automatically generated

Logo

Description automatically generated with low confidenceAs shown the enemy is drawn on the frame and currently appears in the set coordinates (0,0).

I then added in a new function to find the closest pathway tile to the bottom right corner.

Text

Description automatically generated

This function which was created in the GameUI class is then called in the GameUI function within the Menu class.

Text

Description automatically generated

When testing I found that the enemy would inconsistently be spawned In the correct location.

A picture containing building

Description automatically generatedHere it has been drawn on a wall tile.

A picture containing building

Description automatically generatedHere it has been spawned correctly on a pathway tile.

I realised when reviewing the code designed to generate this location that I had accidentally placed the number of rows in the column bracket on my array and vice versa. While testing the spawn location I also realised because of how my algorithm generates the maze that there will always be at least one path tile within a 2 by 2 tile area. This meant I could cut down code that wasn’t required and this allowed me to shorten the function.

A picture containing calendar

Description automatically generated

I next just expanded the existing functions to manage the drawing and placement of all four enemies.

Graphical user interface, text, application, chat or text message

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

­­Text

Description automatically generated  
Qr code

Description automatically generated

As shown by the screenshot I made the enemies different colours for now for future testing purposes so I can easily distinguish which one is which. The enemies where also always drawn in the correct locations however I noticed that in terms of playability there may be an issue with the main character spawning in a random location. Although unlikely it is possible for the main character to be drawn on or next to the enemies which would make starting the game hard for the user. Because of this potential issue I have decided to rework how the main characters spawn position is found.

Text

Description automatically generated

I chose to copy the same method as how the enemy locations are found. The function simply now checks 5 location in which one will be free.

Now I need to create a function which will allow the enemies to move around the maze and attempt to catch the main character. I first considered implementing the A star algorithm for the enemies to navigate the maze however after careful consideration I realised this algorithm would only work if the node layout was predetermined or if I determined close nodes when making its way through the maze. Due to my mazes being randomly generated the program would have to find the nodes and decide what ones connect to each other and in essence by doing this the program has already solved the maze or find viable nodes as it progressed which would require a far more complex algorithm to achieve a tiny bit more efficiency. I decided to go with a process that checked every route and backtracked if a dead-end was met.

Text

Description automatically generated

Text

Description automatically generated

­­Text

Description automatically generated

The find next move function I created made use of some other functions I created to keep the find next move function as simple as possible.

Text

Description automatically generated

Text

Description automatically generated

This first iteration of the function ran into an error where it returned the stack being empty. This was due to sometimes the enemy started in a location which met the conditions to be considered a dead end. This meant the program would try and pop a piece of data off the stack which didn’t exist.

Text

Description automatically generated

I decided to add in an extra condition to check that it wasn’t where the enemy started.

Text

Description automatically generated

I added an AND to check that the tile the enemy was on wasn’t the first tile and if so, it would not be considered a dead end. Even after this adjustment, it was running into situations where it would revert to a back trackable node but because all options from this node had been explored it would cause a standstill where the program just recorded the same location.

Text

Description automatically generated

Text

Description automatically generated

I decided to add in a few Boolean variables which would add in extra requirements that had to be met to satisfy an if statement. This were required to make sure the next move chosen hadn’t already been visited. I also added in many print statements to see how the program progressed as it ran, and this allowed me to follow the execution of the function and spot errors.

A screen shot of a computer

Description automatically generated with low confidence

Due to the issue of the function choosing a backtracking node which had no more options I decided to change how I defined a node within the node() function.

Text

Description automatically generated

The new way I was deciding what contributed to a node made the code less complicated and in turn fixed the issue a bit. The issue was still the same that some nodes were being exhausted meaning I would need to add in a statement to check if a node is exhausted however after the change the program ran successfully for longer. The issue is where coordinates begin to repeat.

Text

Description automatically generated

By this point in programming the pathfinding algorithm I realised how overcomplicated my code had become so I decided to take a step back and reorder the existing code I had. This included things like checking if the current location is a dead-end at the start of the function and not the end and so on.

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

The code was now far less confusing and was able to produce a valid path inconsistently. There were still times it ran into errors, but it seemed a 50 50 chance it would find a path.

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Obviously, it is required it can find a path 100 percent of the time however I realised that from the paths which were found they favoured the up and right direction. Due to my levels having more than one path to the main character they most of the time favoured the path to the right and up. This was because of the order my program prioritised directions however when the user is playing this may make the characters predictable. This prioritisation method also meant that the program almost never found the shortest route and instead it found the longest one. This caused me to reconsider the A star algorithm as the extra complexity to reduce how predictable my enemies were would be worth it.

I initially declared some new variables and created some new functions within the GameUI class.

Text

Description automatically generatedText

Description automatically generated

These new functions were purely for calculations to reduce the repetition of code within the main function.

Text

Description automatically generated

Text

Description automatically generated

I then created a function called potential nodes to find the next possible locations once a new node had been selected. All these functions contributed to the main function findNextMove.

Text

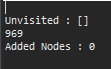
Description automatically generated

A screenshot of a computer

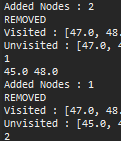
Description automatically generated with medium confidence

I chose to use an ArrayList to represent all my values due to the constant space between two nodes values. For example, if I store the f value of one node within array index 0, the next f value for the next node will be in array index 7 and the next in 14 and so on. This would make accessing nodes easier than using a hash table.

Currently when the program executes the function is able to locate the character occasionally and other times becomes stuck in a loop where it runs out of nodes to visit. In an attempt to fix this issue I have added many print statements throughout the function to keep track of variables. These include the nodes added to the visited array list, the nodes removed from the unvisited array list, the number of nodes added per loop and then just an output of both arrays.



As shown above, the loop is on its 969th iteration and is outputting an empty unvisited list with 0 nodes being added to the unvisited array each iteration. This is an issue as the location of the character has not been found yet so there should still be more nodes to visit. I realised that by finding potential nodes and adding them to the unvisited array that the unvisited array list should be growing larger and larger than the visited array list each iteration. As I had confirmed that the number of nodes being added was always one or larger for each node that is removed.

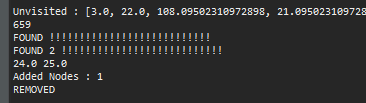


I concluded that there must be an issue when removing a node from the unvisited list. As of currently the condition that has to be met to remove a node from the unvisited list is that the F value must be the same as the node we want to remove. However, through the use of simple mathematics, it can be determined that when using a uniform grid such as the one I am using there will be many points in which the addition of the heuristic and the distance from the starting node will be equal. This will then remove multiple nodes from the unvisited array instead of just the desired one.



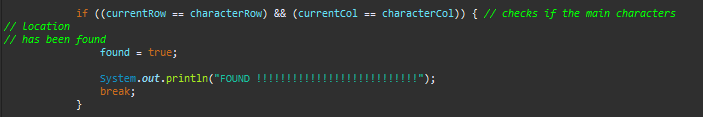
To fix the issue I added an extra condition to be met which would make each combination of these conditions unique to only one node. The new conditions were that the F value is the same and that the row and col were the same as the one we want to remove.

After the correction, It made the program manage to find the characters node each time the function was run.

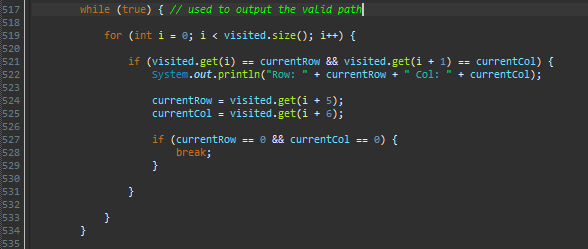


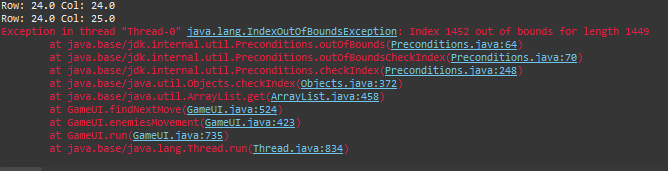
Once found the output was FOUND and FOUND 2. The first found declared that the position of the character was found and acknowledged by the program as the endpoint. The second found was used to check if the location of the character had been added to the visited array but not acknowledged as the endpoint. The only issue now was because of where the branch statements are used the loop carries on through for one more iteration before ending.

This was however an easy fix and just required the addition of a break in the if statement.

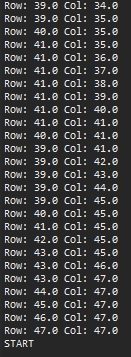
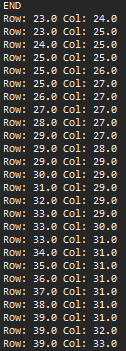


Now that the endpoint (the position of the character) is being found reliably I decided to create a loop to output the path so it could be checked. This is being used purely as a method of testing as when being used by the AI the only node required would be its next move.

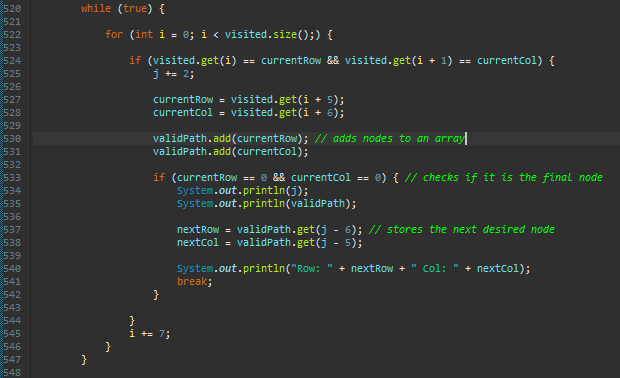




This output method caused an error to occur once executing due to the loop attempting to access indexes out of the bounds of the array. The reason for this was because the integer i was increasing at a rate of 1 per iteration. This meant that every index in the array list was being compared against currentRow and currentCol. However, this is an issue due to previous nodes also being stored in this list. I fixed this by changing it so i increases at a rate of 7 per iteration.

This is the output produced which is each node the enemy would have to move to in order to reach the main character in the shortest time possible. Now that I had completed the function to find the main character this made certain functions redundant such as deadEndReached which I was able to remove.

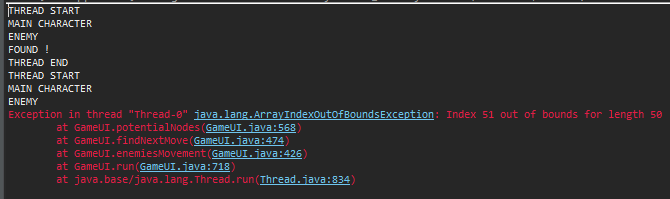
I now want the enemy to make its way towards the main character. I started by modifying the loop which was used to output the valid path.



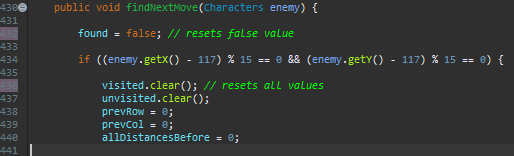
I had to declare a few more variables such as a new arrayList validPath and two new doubles to hold the next desired node.

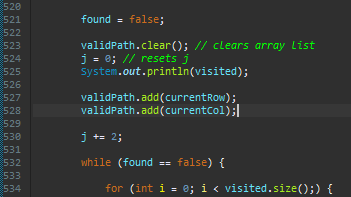


This output produced was what was expected and shows the array of all the nodes that are required to reach the main character and the next node. I found that although the correct path is being found the thread becomes stuck after one or two iterations.



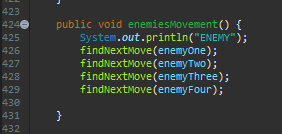
The program managed to reach the second iteration when the valid path to the main character was shorter. Due to the error message being an out of bounds error I noticed that i hadn’t made it so variables reset after each run-through of the function. I made all variables that needed to be specific values when the findNextMove runs reset and I implemented a set of if and elses to allow the enemy to move.



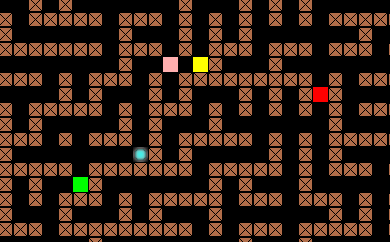


The addition of this code allowed the program to function as intended and the enemy now makes its way to the character. Once the character is reached it continually follows the character when it moves. This shows that the enemy successfully updates its valid path array if the character’s location changes.

I then added a call to the function with each enemy to allow all 4 to move towards the location.



This function still operated mostly as expected when being used for four enemies. The enemies which reached the enemy before the last enemy would oscillate between two locations until the final enemy makes it to the character. The enemies will then all stop on the main character once the last one has reached the location.



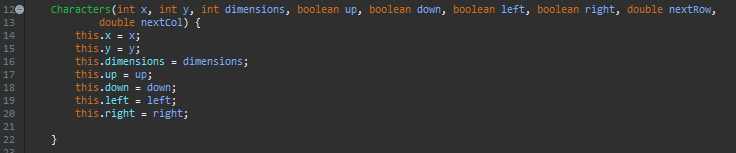
The issue of the enemies oscillating on the character was due to the position of the if branches within my code. They were being called no matter whether a next move was required or not which meant if a next move was not required to reach the main character it would repeat the last move. I managed to erect this issue by moving the ifs which decided the moves into the loop which handled the creation of the valid path. This meant that the loops would only be reached if there was a next move.

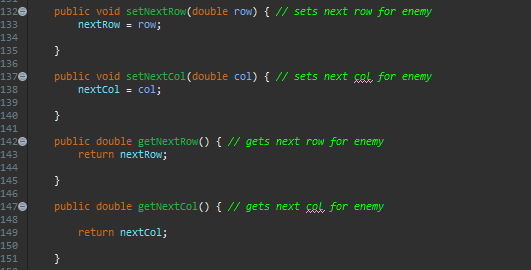


Although the enemy currently can make its way to the character it does it in moves of 15 pixels per 7 milliseconds which is significantly faster than the main character can move leaving no opportunity to escape.

I next needed to make the enemies move at the same speed as the main character to make it possible for the character to evade the enemies. To lower the speed the enemies moved at I had to create some new functions to store and retrieve the enemies’ next move. Due to the enemy having to appear to have smooth movement he would have to reuse the same next move, and this requires for the next move to be stored.

I created two new variables nextRow and nextCol to be handled by the objects in order to store the next move for each enemy. These two variables are also required to create some new functions in order to set and request the new values.





Now that all the required variables and functions have been created I can change where the values are being stored when the next move is found.



Now that the next location has been stored and can be accessed through the new functions created I can freely compare the next move to where the enemy is supposed to be without any concern of the next move changing until it is supposed to. For this to work I moved the if and else statements outside of the loop so no matter if a new next location is to be considered it will always move the enemy.





The enemy’s movement per iteration has now also been changed to 1 which will make the speed of the enemies more manageable to the user to evade and is the reason why the previous changes must be made. This is because the next move is only calculated when the enemy is in line with the grid.

I decided to change the appearance of the enemies as currently, they appeared in a very basic form. I decided to keep their colours so they can be easily distinguished from each other.

Text

Description automatically generated

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

The code used was very similar to how I redesigned the main character and yielded the intended outcome.

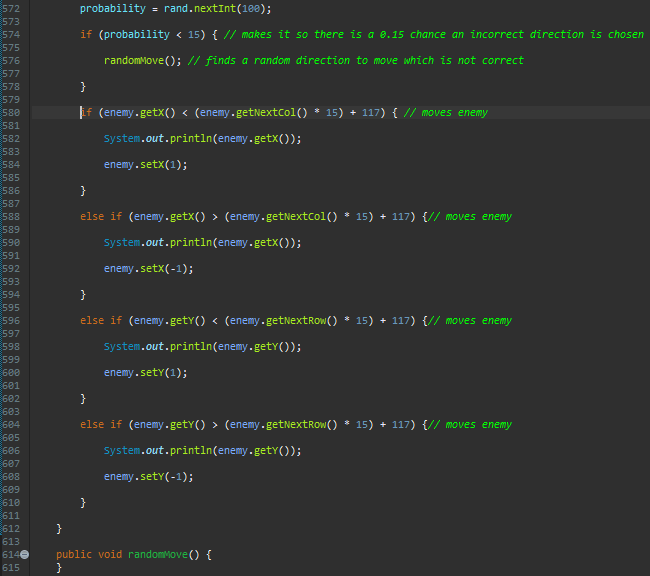
A picture containing building, window

Description automatically generatedA picture containing shape

Description automatically generatedA picture containing building

Description automatically generated

Next, I aimed to make the decision-making of the enemies a bit more complex by adding probabilities that they will choose an incorrect direction. This will then make it easier for the user to evade the enemies due to the map containing many dead ends.



I firstly created a new function called randomMove which would be called if the variable probability is below 15. This means there will be a 15 percent chance that the function will be called and an incorrect move will be made.

I made a critical change in terms of how complex the function would be at the start of making the randomMove function. I initially opted for the array approach for storing the data I would need access to when choosing a move which caused me to run into indexing errors.

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

However, I quickly realised with a need for a varying array length it would be more suitable to use an array list. This also meant the variable placeholder variable could also be removed.

Text

Description automatically generated

I then added a switch statement to decide what direction the character will move where the probability it will move in each direction is equal.

Text

Description automatically generated

However, when running this new code, I noticed certain enemies were moving through walls when attempting to move left and right.



I noticed this was because I was calling this new randomMove function every iteration meaning it was being used when the character wasn’t perfectly in each grid location but instead between two. Due to java then choosing to round in the calculations used this made the enemies think they are in locations in the maze that they are not.

The fix for this issue was to simply move the if statement which is called the randomMove function into the if statement which was only used when an enemy was in a grid spot.

Text

Description automatically generated

On the next test, I chose to raise the probability that the enemies would choose an incorrect direction to 50 percent purely to check the function was working as intended. The enemies chose incorrect directions based on this probability however I noticed their timing on when to choose an incorrect direction didn’t necessarily make sense.

For example, if the enemy is moving down a ‘corridor’ where its only choice of direction is up or down it doesn’t make logical sense to randomly change direction halfway down this corridor as the enemy has not been faced with a logical choice for directions. This illogical choice from the enemies was due to the randomDirection function running when the enemy reaches each new grid spot. However, I would like it to appear to the user as if the enemy is consciously considering each direction instead of erratically changing direction.

To fix this issue I first added a new if statement to count how many potential moves there are and if there are 3 or 4 potential paths the randomMove function will be called. This means all the random moves will be made at junctions making the movement less erratic.

Text

Description automatically generated

Although this fixed the issue with the character making wildly illogical movements it only reduced the erratic movement slightly due to the incorrect direction only being used for one grid place movement and then correcting itself.

My solution for this issue was to make the character move in this random direction until it collides with a wall or reaches another junction.

Text

Description automatically generated

I created a Boolean variable called incorrect path so that if an incorrect path is being used it will bypass the process in which a next move is chosen and continue in this direction.

Text

Description automatically generated

I then altered the if and else statements to store direction being travelled for reference in later operations.

Text

Description automatically generated

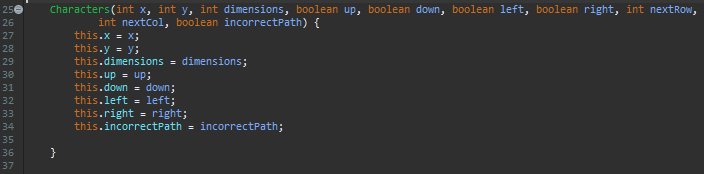
The direction recorded earlier is then to be used to alter the next row and next col so that it continues to move in the same direction. The branch statements all consider what direction the enemy is moving, if a move in that direction is valid and that it is not on a junction. Due to me having to use a junction finding operation multiple times I also decided to make this a function.

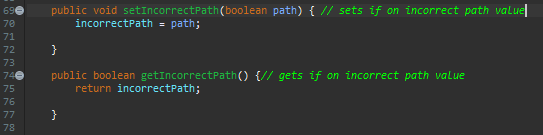
A screenshot of a computer

Description automatically generated with medium confidence

Now when testing this new code there was an issue where all the enemies would remain still in a certain position. This occurred consistently after a varying number of iterations of the loop within the thread.

I firstly noticed that the variable incorrect path was being used for all enemy objects meaning that when one enemy changed its value it would then incorrectly influence the next enemy. To fix this I added another variable to the character objects to store this value for each enemy. I created corresponding get and set methods for this variable.

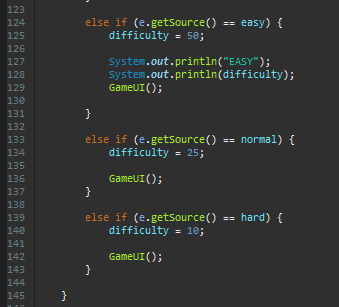


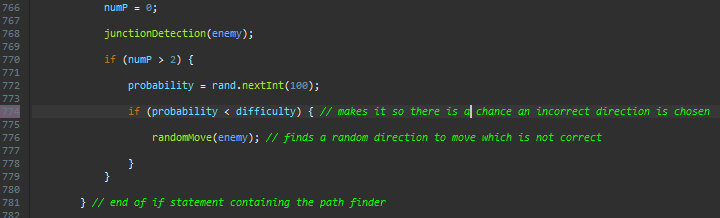


Although this was a valid error that had to be fixed it did not erect the issue where the enemies were stopping after random amounts of time. I found that the mistake causing this error was simply that I was incrementing the wrong variable when trying to change the next location to be used by the enemy.



I changed currentRow and currentCol to enemy.set(enemy.get…); functions and this fixed the issue of them stopping after a given amount of time. I next wanted to make the usage of this function be based off of the difficulty chosen. I attempted to cause the choice of difficulty to influence this function by simply assigning a number to the difficulty button chosen and then being used in GameUI.





The amount an incorrect direction is chosen is now influenced by the difficulty chosen. However, when I attempted to select different difficulties the difficulty variable was equal to 0 in GameUI. I decided to implement more print statements for testing purposes.

As shown in the console output when the difficulty Normal is selected the difficulty variable is assigned the value 25 however when it is used in the GameUI class it is equal to 0.  
This issue was due to the difficulty variable not being set to static in the parent class which is Menu.



As shown by this console output the third output is now 25 which is the print statement within the GameUI class which shows the variable is being inherited by the subclass correctly.

Now that the difficulty variable was able to pass its value this added more functionality to the menu buttons which allowed you to choose a difficulty. To make sure there was a significant difference between the difficulties I ran some tests which consisted of me timing the enemies. I first found the average time it took all the enemies to reach the character while he remained still.

Easy- 52.228 seconds

Normal- 27.844 seconds

Hard- 25.88 seconds

As expected, there is a large difference between easy and normal difficulty however not so much of a difference between normal and hard. Due to the game-ending after being caught by the first enemy to reach the main character I also calculated the average time for the main character to be touched by an enemy.

Easy- 21.78 seconds

Normal- 17.39 seconds

Hard- 13.942 seconds

Although the map is randomly generated, and a more accurate estimate would be made after thousands of executions this data gave me an idea of how the difficulty ranged from easy to hard. I decided not to make any changes to the value the variable difficulty was set when a button was selected due to a consistent gap of time between each difficulty as shown in the second set of test data.

Evaluation

Stage 6 had a focus on the development of the enemy objects ranging from how they would appear to how they navigated the maze. I managed to meet all the goals I had set at the start of this stage with a couple of major setbacks when programming the solution. These included converting the depth-first search into a variation of the A star algorithm and redesigning how a spawn location was found for the main character. Each search algorithm I made came with its fair share of out-of-bounds errors and mathematical imperfections however these were not too hard to correct, and the main issue was the time required to reprogram how the enemies navigated the level. The variant of the A star algorithm I decided to use did not check that the route was the shortest at the end of the function when the main character had been reached. This was due to the distances between nodes being equal, the final heuristic being the same for all routes and the next node to be checked being chosen because it is on the shortest route meaning when the main character was reached that was the shortest route. I decided to mainly focus on getting the enemies to locate and move towards the character before considering the enemy design as this was a less important and easier task. When programming stage 6 I noticed that breaking the character’s class down into enemies and main character subclass as shown in my UML diagram was not worth it as they shared all the same variables and all that differed was how the variables were used although some new methods were declared for the enemies the class was still at a manageable size.

## Stage 7

* Add Items in the Game
* Create a collision detection system for the main character, items, and enemies
* Change the appearance of the items

I first started by creating a subclass of game objects called Items and adding a constructor. I noticed when creating this class which extends GameObjects I had not created a constructor within game objects and decided to. This meant I had to make some minor changes to the Characters constructor.

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

I next created four items objects and assigned them values. I also created two methods to be used when setting the location of an item to check if it is already on the board or if it has to be replaced as when the main character collects an item it should reappear somewhere else on the level.

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

After the creation of these methods, I made a function in the class GmaeUI to find the positions where the items will be on the level. I created two functions called items and itemPosition. Items is called in the thread as a main function which means when this function is called it will then handle everything that the items need.

Text

Description automatically generated

Text

Description automatically generated

However as shown above I found that when attempting to use setX() as I previously have with the characters it did not work. This was due to me having not made these methods accessible to the items subclass as these functions were defined within the characters subclass and not GameObjects. To fix this issue I moved the functions to the GameObjects class.

Text

Description automatically generated

Text

Description automatically generated

To check if this function worked, I made the program draw simple magenta squares where an item would be.

A picture containing qr code

Description automatically generatedThe function selected locations as intended on the level however due to the function changing the number in the mapArray to a 4 in order to make sure more than one item is not placed in the same place this meant that the main character and enemies would recognise this as a block they cannot travel on causing a standstill if they were blocking pathways.

A picture containing qr code

Description automatically generatedBecause this meant major changes would have to be made to how the enemies and main character methods operated, I decided to opt for a long if statement to compare each location of each item.

Text

Description automatically generated  
The if statement worked as intended and prevented multiple items from being placed on each other while also allowing the main character and enemies to move freely.

Next, I needed to develop the collision detection system for the enemies and the items. I decided to do both detection functions at once due to how similar they are. The development of these functions would also allow me to test the previous itemsPosition method.

A screenshot of a computer

Description automatically generated with medium confidence

At the moment the enemy collision detection only checks if an enemy is colliding with the main character and outputs a statement to the console saying it has collided. The item collision detection does the same as the enemy one for the items however it also makes sure the item is replaced on the level when an item is collected.

Text

Description automatically generated

I then made it, so they are called at the start of each iteration of the while loop within the thread. Now that this had been completed, I could test the new collision functions and the previous intemPosition function.

For this test, I set the thread sleep timer to 200 to allow me to see if the message is output instantly on collision.

Text

Description automatically generated

A picture containing building, different, colorful

Description automatically generatedText

Description automatically generated with low confidenceAs shown here when an enemy is colliding with the main character an output of collision 1 is sent to the console.

A picture containing qr code

Description automatically generated



As shown here the collision detection also worked correctly for the items however as shown in the maze image only 3 items are present. This means the collected item has been placed in an incorrect location or simply hasn’t been redrawn. In an attempt to rectify this issue, I placed two print statements within the function to output where an item is placed and a repaint to check it was being redrawn.

Text

Description automatically generated

The repaint had no effect on the result as the item was still not present however I found out the items position was being set to off the screen.

Text

Description automatically generatedThis output confirmed there was an issue with the replacement of the item. I reviewed the function I found that I hadn’t considered how the functions .setX and .setY worked. I forget I set them to mange increments in numbers. For example .setX(1); would increment the X position by 1 each time it is used. To fix this I simply just set the X and Y coordinate to 0 before setting the actual position of the new item.

Text

Description automatically generated

Text

Description automatically generatedA picture containing qr code

Description automatically generated

The new code allowed an item to be collected and successfully placed in a valid position on the map.

Now that the items are appearing correctly, I need to change their appearance. I achieved this the same way as I implemented the enemy and main character images.

Text

Description automatically generatedGraphical user interface, text

Description automatically generatedText

Description automatically generated

This yielded an output with the same functionality as before, but the items had a different appearance.

A picture containing light, window

Description automatically generatedAs shown the items are blue and share a common colour scheme with the main character to make it clear what the items are when trying to differentiate them from the enemies.

Review

Stage 7 was focused on the implementation of items within the level and how the enemies and items would detect a collision with the main character. This stage was completed quickly due to its low complexity. There were instances where errors occurred due to classes having to be reformatted using super() and some methods had to be moved into the superclass to allow access to all subclasses. There were also a couple of logical errors where numbers were not being stored/manipulated correctly however these proved to require only trivial solutions. Overall, the completion of this stage was smooth and simple.

## Stage 8

Goals

* Add a score counter which is displayed at the top
* Add a timer which is displayed at the top
* Develop the collision detection to end game when colliding with an enemy
* Allow the user to retry once the game has ended

I firstly implemented all the methods and variables needed to total the score.



Text

Description automatically generated

The get score function was made within the item’s class and used to fetch the score of whatever items are being collided with. Once these had been declared I could adapt the collision method to make the game total the score as the items are collected.

Text

Description automatically generated

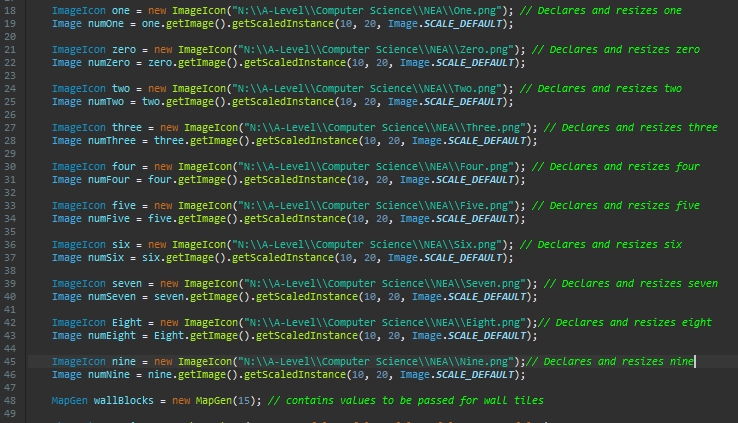
I then implemented an if statement to output the total score when an item is collided with to check it was being totalled correctly.

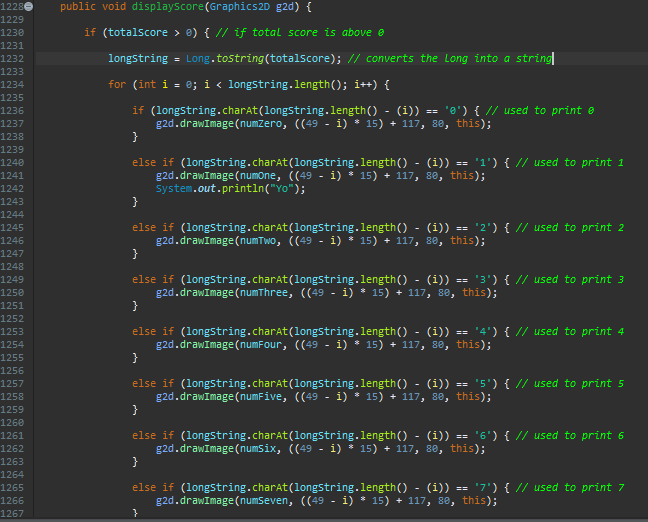
Text

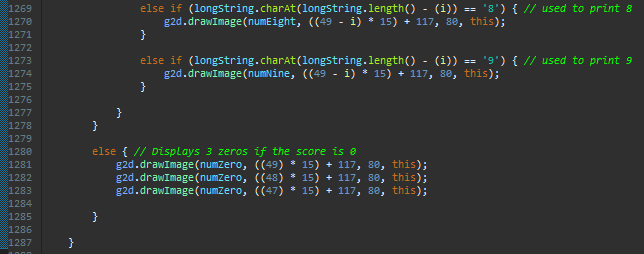
Description automatically generatedAs shown by the console output the score was being totalled correctly and added to after each collision with an item.

After I had confirmed that the score counter was totalling the score correctly I had to implement a display function so the user can actually see his score. I decided that I would used images to keep the pixelated retro style of the game consistent with the score counter.

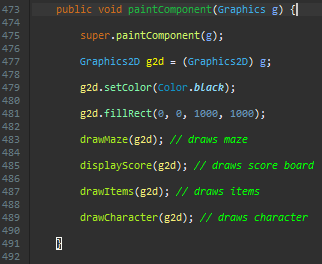
I first declared and resized all of the images I was going to use.

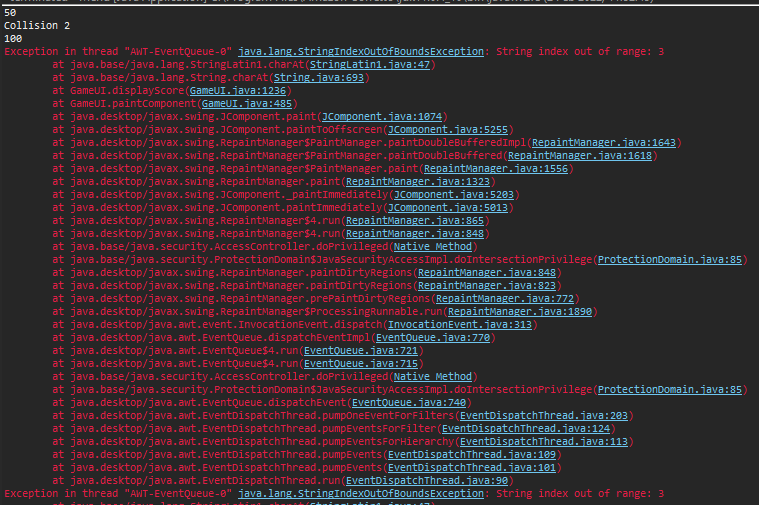
  
I next needed a function to take the total score and split it into individual integers to be output through the use of images.



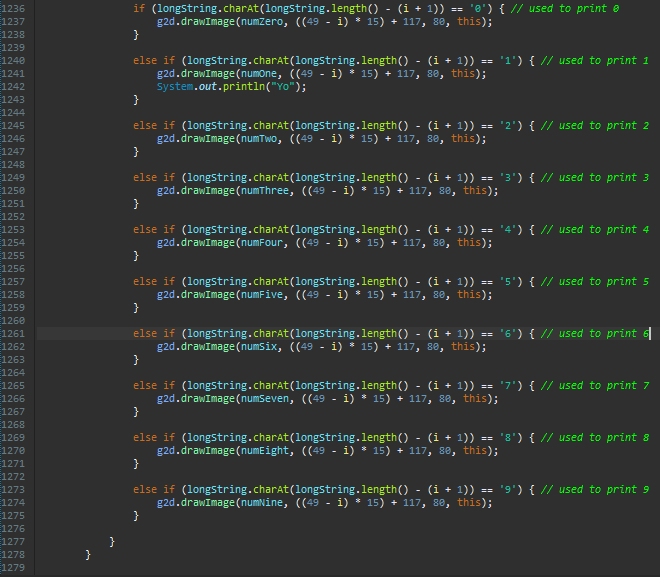


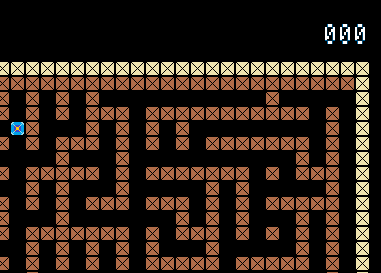
The function converted the long to a string and then compared each char to its corresponding number and then draws the images in order. If the score is 0 the function will draw 3 zeros which I decided to do so the user knows where the score will appear.

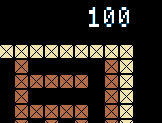
I then made it so the function is called through the paint component function so the score will be drawn on the frame. I decided to run the program to test if it functioned as intended.

 The game ran smoothly until the main character collided with the item which caused errors to be output. The main error was an out of bounds error which then had a knock-on effect causing many other errors.

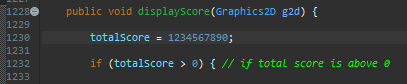
To fix this error I checked the function which manages comparing the chars with the corresponding numbers.

  
  
To fix this issue I added one to I before taking it away from the string length. This was required because charAt() indexes from 0 however .length() indexes from 1 so to remove the difference 1 need to be added.

Once I had changed this I tested to see if the program would run correctly. As shown in the screenshot the first three zeros when the score is 0 are shown correctly in the top right of the maze. The score was also correctly shown when items have been collected.



I also chose to change the number the totalScore variable is holding to present all the numbers to check their in-game appearance and I did a check to see what the maximum number of integers that could be displayed are.

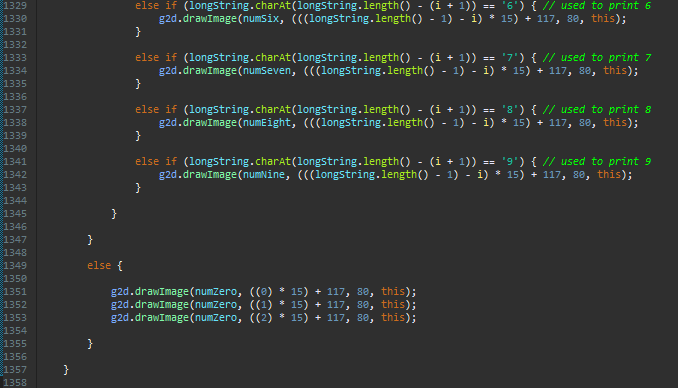
The first test was successful as each number was clear and easy to read while still maintaining a retro style.

The maximum score is 999999999999999900. I decided that this was a high enough limit for the score as achieving this would be almost impossible for a human. Assuming each item is a score of 100 and it takes on average 5 seconds to get from one item to the next it would take them 15854895.9919 centuries to reach the score limit.

I next decided to implement a timer in the top left of the screen to enable the user to see how long they have survived for. Due to the game having varying difficulties I decided that a timer would be a useful addition as the time survived should be just as important to the user as the amount of time they managed to survive for.

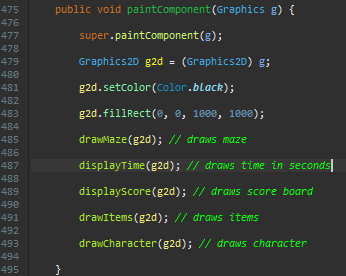
I firstly declared a new variable named startTimeSeconds which store the start time in seconds so it could be used in calculations later in the method.

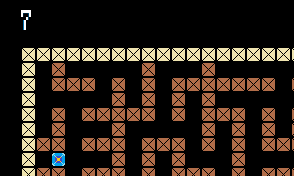
  

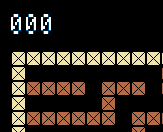



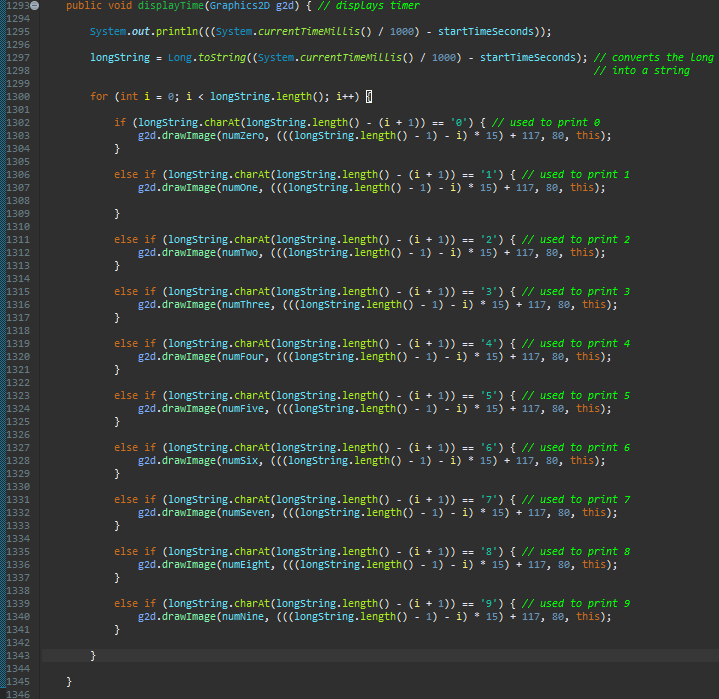
I then created the displayTime function in order to make the time visible to the user. It works in almost the exact same fashion as the displayScore function keeping the art style consistent and the only change is the position the numbers are placed in. This is in order for the timer to display in the top left instead of the top right. I found that the variable longString was able to be recycled and used again in this function as it served the same function. The time was able to be calculated by taking away the start time from the current time.

Now all that was left to do was make my program paint the timer in the top left of the interface.

I simply called the function from the paintComponent method and passed g2d. I then chose to run the program in order to check if the timer was displayed correctly.

 The timer appeared in the correct location an incremented by one every second as intended however I noticed that I have coded the function so that when the time is zero it outputs 3 zeros. This worked for the score as it increments by 100 at a time however due to the time using single digits it appears as a glitch or bug when it goes from 3 digits to one.

This is the output when the time is 0. This output was not ideal due to the issues I mentioned above so I decided to change it to a single 0. I changed the function so it no longer had an if and an else statement but simply just output the value being stored in the long.

This was done because there was no longer a special condition where the long had one integer but three had to be output for aesthetic reasons like in the score function. By doing this it provided a smoother transition between the numbers 0 and 1 at the start of the counter.  
  
  
The next step in stage 8 was to add more functionality to the collision detection so when the main character collides with an enemy the game actually ends.

I decided that to achieve this I would create a Boolean variable called gameEnd that would be checked on each iteration of the while loop within the thread. The value of gameEnd would be influenced by whether the enemies have collided with the mainCharacter.



The variable was declared within the menu class as it would be required by both the Menu and GameUI class.

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

I then proceed to implement where the variable is changed due to a collision with the enemy and how the game is stopped when the collision occurs. I also chose to move the enemy collision detection method calls to the end of the while loop so they are the last method to execute before the loop is broken out of. This then called the GameUI method from the menu class which had been altered slightly.

I initially attempted to move the declaration of the thread and GameUI object to the top of the class.



However, when running the program to check this caused no major change in functionality a long error statement was output suggesting there was an issue with an infinite recursion occurring.

Text

Description automatically generated

When attempting to identify the root of the issue I realised that my knowledge of the implementation of runnable and paint function are not versed enough to resolve this issue efficiently.

I decided to move the declaration of the two variables back to their original location and attack the problem from a different angle.

Text

Description automatically generated

I made use of branch statements in order to make sure that another of the same frames was not created and the else can be used to construct the end game screen once the user has been caught.

I then changed the size of the frame within the else statement. Although this was not the desired size, I wanted to check the else statement was successfully reached and a change was made.

Text

Description automatically generated

Qr code

Description automatically generatedGraphical user interface, text

Description automatically generated

As shown by the output the else statement was successfully reached and the print statement was output but the frame size had not changed. In a series of tests, I attempted other trial and error solutions as I was unsure of the origin of this issue.

Graphical user interface, text

Description automatically generatedI attempted to close the frame altogether as although this was not the desired output the frame should have no issue in closing and should not be restricted by any other methods.

This also yielded the same output.

Graphical user interface

Description automatically generatedText

Description automatically generatedI next attempted to make the frame visible as I was clearly unable to manipulate the frame, I intended to change so I assumed another frame was being created. This was true as when I set the frame to visible another frame of the set size was being displayed. I found that this was due to an instance of the frame being created when the paint method was being used and I, therefore, would have to make the JFrame variable static.



Doing this caused the frame to be able to be changed again and as shown the size was successfully changed.

A picture containing text, clock

Description automatically generatedHowever, although changing the variable to static was a step in the right direction, I didn’t intend to change the size of the window but instead change the appearance of the window to a blank screen displaying the score achieved, time survived and a game over text.  
Text

Description automatically generated

I managed to achieve this be adding in some branch statements into the paint component method which checked what to paint based of if the game has ended. This method yielded the correct output.

Shape, rectangle

Description automatically generated

When attempting to add the title bar which contained the name of the game onto the screen, I found it didn’t appear, so I tried changing the title bar variable to static and it worked.



Text

Description automatically generated  
Graphical user interface, text

Description automatically generated

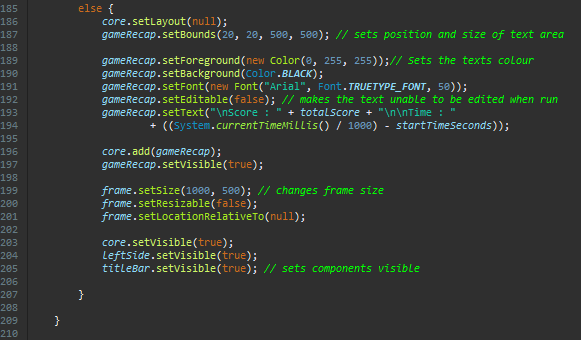
After understanding that many of my variables may have to become static to be used as I intended, I decided to make these changes. I also decided to change methods and variables to private, protected, or public to make debugging easier as debugging for a couple of the last errors has become more time-consuming due to complications with variable access.  


Through doing this process I was also able to identify many unused variables I had not noticed to remove as shown by the yellow lines. I changed all variables which are only required in the class they are declared as private, and the others protected as these can be accessed by all subclasses of a package. I also did the same for all methods and functions.

I next realised that blending painted graphics and my JComponents when creating the end game screen would become complicated, so I chose to leave the else in paint component empty and move my end game code into the menu class.

Text

Description automatically generated  
I moved the start time variable and total score variable into the menu class so they could be used in the new JTextArea which I declared as gameRecap.



I then used these new variables to create a new end screen which displayed the score and time along with allowing you to select another difficulty so you can play again.

This yielded a successful output of:





I also had to reset each variable representing the score, time, and whether the game has ended so that it runs smoothly when the game is replayed.

Review

During stage 8 I set out to add a timer, score counter, end game menu, and develop the collision detection with enemies. The issues I ran into in this section were largely due to iterations continuing out of bounds, infinite recursion when creating and declaring objects, and instances of variables being created when not needed. These issues were quite easy to resolve and just required minor maths changes or declaring variables as static. I also proceeded to make variables private, protected, or public depending on where they are used. This should have been done as I completed my project as it would not be manageable to do it all at the end in a large project however due to my project being relatively small this was not an issue. Doing this also helped me with testing.

# Part D

## Post Development Testing

My first post-development test is that the levels are generated randomly and are different each time the game is played. I am going to test this by selecting different difficulties and replaying the game to verify that the maze is different.

1. Selected Easy from the main menu.

Qr code

Description automatically generated with low confidence

1. Selecting Easy when the user wants to retry after a loss.

A picture containing qr code

Description automatically generated

1. Selecting Normal from the main menu.

Qr code

Description automatically generated

1. Selecting Hard when the user wants to retry after a loss.

A picture containing qr code

Description automatically generated

As shown the level is generated randomly each time the game is played as intended. Due to the three different buttons all calling the same level building function this makes sense.

The second test was to check the character collision detection. This includes collisions with enemies and items in which a collision with an item should raise the score and a collision with an enemy should end the game.

1. Pressing up into a wall



1. Pressing right into a wall

A picture containing building, window

Description automatically generated

1. Pressing down into a wall

A picture containing building, door, window

Description automatically generated

1. Pressing left into a wall



As shown in the four tests the main character successfully identifies an invalid move when a direction is selected in which there is a wall blocking it. The program successfully checks the coordinates of the main character and validates its position in correlation with the level array.

1. A collision with a single item

A picture containing text

Description automatically generated

As shown after a collision with a single item the score increments by 100 as intended and the game continues.

1. A collision with an enemy

A screenshot of a computer

Description automatically generated with medium confidence

After a collision with an enemy, the game returns to the end of the game screen in which the three difficulties are presented in the form of buttons where the user can choose to retry the game. This was the desired result after a collision with the enemy as the collision is detected correctly and the user gains the option to retry.

The next post-development test was to check that the score counter is working correctly. This includes adding up items correctly, displaying correctly, and showing the correct score after a game has been restarted. I will test this by navigating the maze and collecting items while colliding with and not colliding with enemies.

1. Collecting 3 items normally, one by one while not in a collision with an enemy.

A screenshot of a computer

Description automatically generated with medium confidence

1. Collecting 3 more items after the user has selected to retry by choosing another difficulty.

A screenshot of a computer

Description automatically generated with medium confidence

1. Collecting an item while colliding with an enemy.A screenshot of a computer

   Description automatically generated with medium confidence

As shown in these first three tests the score is added up correctly and displayed correctly after the game as required. The second test verified that the score counter is reset after each turn and each test shows that the scores are added correctly. The third test verified that it doesn’t matter if a collision with an enemy and item occurs at the same time as collisions are detected from coordinates which are then compared to the position of each object before the game ends.

I now plan to carry out the first two tests again however verifying that the score is displayed correctly in the game.

1. Collecting 3 items normally, one by one while not in a collision with an enemy.

A picture containing text, building

Description automatically generated

1. Collecting 3 more items after the user has selected to retry by choosing another difficulty.

A screenshot of a computer

Description automatically generated with low confidence

The final two tests showed that the score is displayed correctly in the game regardless of whether the game has been restarted or if it is the first game being played.

The third post-development test is to check the movement of the character while the game is being played. This will check that the character and enemies do not violate the rules of the game such as passing through walls.

1. Pressing a direction once to travel down a straight path

A picture containing building

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

In test five the direction being travelled was upwards and the up-arrow key was only pressed once. The main character travelled in this direction until a wall was met and stopped which was the intended result.

1. Attempting a string of imperfectly timed presses. For example, pressing up when it is not a valid direction to be taken at a given instance in time.

Shape

Description automatically generated

Shape

Description automatically generated

The main character can be manoeuvred through the level using the arrow keys as shown in test 6. The string of button presses used: Right, Down, Right, Down, Left, Down, Left, Up, Left. These 6 tests demonstrate the manoeuvrability of the main character and how no basic laws of the game are broken.

The next test was to check that the detection of the main character carried out by the enemies was correct. This test was carried out to make sure that the characters are not just moving randomly and have a destination to reach.

The same path-finding algorithm was used for each difficulty with the difficulty only influencing a variable with no impact on the pathfinding algorithm. Due to this, I decided to test this on hard difficulty as using this will yield the clearest results.

1. Enemies moving towards the main character on hard mode.

Qr code

Description automatically generatedAs shown by the screenshots on the left the enemies start in their respective corners with the main character in the middle. The enemies over time then proceed to make their way towards the main character until he is reached.

A picture containing qr code

Description automatically generated

1. Enemies moving towards the character on hard mode and the main character has been moved from his starting position.

Qr code

Description automatically generated with medium confidence

As shown by the screenshots on the left even once the characters location has been changed the program recalculates a valid path to the main character and proceeds to follow it.

Qr code

Description automatically generated

The next post-development test was for the instructions. This test will test the functionality of the instructions button, how the instructions are displayed and how the return button functions on the instructions screen.

1. Pressing the instructions button once and then the return button once.

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

As shown above the first test I successful as if the instructions are accessed and used as intended the program retains functionality.

1. Pressing the instructions button, the return button, and then the instructions button again.

Graphical user interface, text

Description automatically generated

The result of this test includes the correct display of the instructions as intended and shows the instructions can be accessed as many times as required.

1. Pressing the return button, the second time the instructions are accessed.

A screenshot of a computer

Description automatically generated with medium confidence

As shown here the display returns to the menu as intended with no errors. The buttons can be pressed as many times as desired without failure due to how the frame is manipulated.

The final post development test will be a test to verify that the difficulty chosen has an impact on the difficulty of the game.

This test was completed by timing how long it would take for the first collision with an enemy to occur without moving the main character.

Difficulty

The final post-development test was to check each difficulty altered the game sufficiently and had an impact.

I chose to test this function by measuring the time it took for the enemy to reach the main character by calculating an average after 5 tests.

1. Easy (All enemies reach the main character)

Average- 52.228s

1. Normal (All enemies reach the main character)

Average- 27.844s

1. Hard (All enemies reach the main character)  
   Average- 25.88s
2. Easy (One enemy reaches the main character)  
   Average- 21.78s
3. Normal (One enemy reaches the main character)  
   Average- 17.39s
4. Hard (One enemy reaches the main character)  
   Average- 13.942s

## Usability Testing

Buttons

Graphical user interface

Description automatically generatedChart, treemap chart

Description automatically generated

Text

Description automatically generated with medium confidence

 implemented buttons into my game to reduce confusion when navigating the menu. This limited the amount of influence that the user had on the game as they could only use pre-validated inputs in the menu so the program wouldn’t crash.  I chose to implement the buttons using a reasonably sized font so it is easy to read for the user and I made it so the button’s font would change colour when being hovered over. This made it more apparent for the user that it was a button that could be pressed and therefore increased usability. The buttons followed a very basic design which stuck to the colour scheme of the game making it more aesthetically pleasing.

Chart, treemap chart

Description automatically generated As shown the easy button is being hovered over in this image and has changed to a blue colour and once the mouse leaves the proximity of the button the text turns white again.

The buttons were very effective as they provided some personality to the menu while also being an extremely effective way of navigating the menu. It reduced the chance of an error occurring to 0% due to a user input and was used to effectively manipulate the appearance of the frame. The use of buttons made it extremely clear how the user was intended to browse the menu and was very easy to use.

GUI

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

Qr code

Description automatically generatedI decided to use a simple GUI to present my game to the user and provide an easy way for the user to interact with the program. I chose to make the size of the window fixed to keep the proportions of each panel within the frame constant to maintain an appealing appearance. The sizes of the frame changed when playing the game to make the game easier to view for the user. The appearance of the GUI was very simple to make it easier for the user to navigate the game. However, the GUI came with a shortcoming which was due to it being a fixed size the in-game UI sometimes was too big for the screen depending on what display it was played on. Although not shown clearly by the screenshots the game was not a Fullscreen game to maintain ratios. Although it was designed for the average-sized display and any bigger this would reduce the usability and accessibility for some users. Overall, this feature was not overly successful because as mentioned it reduced usability for some users vastly while being fine for others. This could be improved by calculating the size of objects within the game based on display size so the game could be played on any sized display. This was something that wasn’t considered during development and could be implemented with an overhaul of most of the calculations within the game. This is because the game carried out most calculations assuming the frame size is fixed.

Font

A screenshot of a computer

Description automatically generated with medium confidenceText, logo

Description automatically generated  
Graphical user interface, text

Description automatically generated

When determining how to use the font to increase usability I chose to stick with a plain font that was easy to read and understand. The size of the font varied depending on what it was being used for however it was readable in all cases. The text consisted of two different colours blue and white which followed the colour scheme of the game. For changing variables such as the score and the time I made sure that the limit of these variables would be reached before the whole number could no longer be displayed by limiting the font size. This feature was very effective at conveying the information to the user easily. Due to mac pan being a retro-style game I felt that the simplicity complements the aesthetics of the game.

Images

Shape

Description automatically generated

  
My program made use of many images to improve usability and appearance. These ranged from using images to acquire the desired font to using images to be drawn as my graphics. These images were used to maintain the retro feel of the game along with helping the user play the game by presenting the score and time using pixel-styled numbers. The use of images in my program was very effective as a usability feature as they managed to bring attention to parts of the menu and game, which I wanted the user to notice. This helps the user play and track their progress with ease.

Layout

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

The layout of the menu and game was designed to be simple to use and seem familiar to the user. For example, the close and minimise buttons were kept in the same place, the return button was placed at the end of the instructions and the title was located at the top of the screen. The familiarity that the layout provided allows the user to navigate the menu easily and efficiently. The only drawback to this layout was that it couldn’t be resized as mentioned earlier meaning if a user wanted a larger window that was not possible with the current design. Overall this feature was quite effective as although it could not be resized it provided the user with familiarity and outlined all the main parts of the menu needed by the user in a way that is easy to view.

Colours

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

I chose to use colours that matched nicely with each other to improve the appearance of the program while also using colours that sharply contrasted such as light blue and black. This makes text easier to read and buttons easier to identify. This increased usability making the instructions easier to read and understand. While in the game the maze uses some colours, which contrast the main menu’s theme of black and blue. This decision was made because I found that the original white and light blue maze became quite taxing on the eyes whereas some gentler colours such as brown made it nicer to play. This also allowed me to make the main character the same colour as the theme of the game without it clashing with the colours of the level. Overall, I think this feature was effective as the colour scheme made the menu and game easy to view as complementary colours were used. The use of a limited number of colours within the menu also maintained the simplicity of the menu making it less overwhelming for the user to navigate.

Instructions

Graphical user interface

Description automatically generated

The instructions were designed to improve usability as they were extremely simple, and they presented every bit of the information the user would need to play the game. This reduced confusion as the user could enter the game with an idea of what to expect. The instructions included a very clear return button as well so the user could navigate out of them when they are done. Overall, the instructions are very effective as they are in chronological order and very easy to read. There is only one improvement that isn’t devastating that it isn’t there but some images of what the objects look like when playing could be useful. Although the objects such as items are very self-explanatory once the game is played once this could help first-time players.

## Success Criteria

Random Generation

This was my first success criteria which were to make sure the levels were randomly generated each time the game was played.

As shown by the four-level generations is different. This is also shown throughout my testing in Part C.

This criterion was: MET

Qr code

Description automatically generatedQr code

Description automatically generated

Qr code

Description automatically generatedQr code

Description automatically generated

Score Counter

The second piece of criteria to be met was that a score counter was present within the game.

As shown below the score counter appeared in the top right of the level and could represent each number as required.

This criterion was: MET

Graphical user interface, application

Description automatically generatedA picture containing text

Description automatically generated

Difficulty Settings

The third piece of criteria to be met was that a selection of difficulties was present within the game.

This criterion was: MET

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

As shown, there were 3 difficulties implemented which when selected passed a value that would be used in the calculations that included a probability aspect within my game.

The user can navigate the maze

The fourth piece of criteria was that the user could navigate the maze.

Shape

Description automatically generatedThis criterion was: MET

Shape

Description automatically generated

As shown a string of inputs has been entered to move through the level.

The enemies can navigate the maze

The fifth piece of criterion was that the enemies could effectively navigate the level towards the main character.

This criterion was: MET

Qr code

Description automatically generated with medium confidence

As shown the following images were taken from the start of the game and about 15 seconds into a game showing the enemies can locate the user when they have moved.

Qr code

Description automatically generated

There is a timer

The sixth and final piece of criteria for my game is the timer which allows the user to see how long they have spent in a game.

This criterion was: MET

A picture containing text

Description automatically generatedGraphical user interface, text

Description automatically generated with medium confidenceDiagram

Description automatically generated with low confidence

The timer appeared while the game is being played in the top left and after the game has ended with the score.

## Limitations and Maintenance

In terms of maintenance, my program could operate correctly forever without any data changes or development as it is all operated on internally and only relies on a working system to operate. However, although not crucial to the operation of the game some maintenance could be performed in the sense of keeping the game up to date by patching errors found by players and adding new content to make the game interesting.

Limitations of my project include-

* Simple Graphics: The graphics within my game are extremely simple and pixelated. The pixelated graphics add to the retro style of the game however the choice to use this style was due to me having to create the graphics. The game also featured very limited animation so the game may feel a bit dry in some places such as picking up an item just causes it to reappear somewhere else and being caught instantly triggers an end screen.
* Shallow enemies: Although the enemies in my game were by no means easy to evade as they operated efficiently, they did become a bit predictable. I feel that a major limitation is how each enemy shares the exact same personality which is head straight for the user. This brings a sense of repetition to the game as you can predict which way the enemies will move easily.
* Level Generation: Levels were generated randomly within my game to keep it refreshing for players who play the game a lot. Although this method of generation is great for addressing this kind of player, I found that all the levels can seem to merge into one as they are so random that no level really has its own personality in a sense that once a level is complete it Is gone.
* Customization: I found that after playing the game for a while if you aren’t into playing the game for the competitive gameplay players quickly lose interest because you are effectively only collecting score to beat your last score. I found that this was not satisfactory for some players so the fact that score can’t be used for things like customization of your character is a major limitation.

## Further Development

* Simple Graphics: In further development, I would improve this by making the levels smaller so each object can be bigger so more detail can be added to each of them. I would also use a digital artist to design the objects within my game as this would be too time-consuming if I wanted a worthwhile output. With these more detailed objects, I could also create some worthwhile animations to immerse the user in the game more.
* Shallow enemies: In further development, I would give each enemy its own personality for example the green enemy might favour a rotation around the level where it visits each item, or the pink enemy might try and cut you off instead of just following you. This would add more depth to the game as a player because you would have to think about the individual reactions of each enemy.
* Level generation- I think in further development this would be an easy problem to fix as I could keep the random generation as an extra game mode so that I can develop custom pre-thought-out levels which the user can complete first. This would fix the issue of each level seeming too similar and still maintain a game mode for dedicated players, so they don’t get bored.
* Customization: In further development, I would improve this by introducing a shop. This would be a location where users can spend points which would be their total score achieved from playing. This would include different enemy, character, and menu skins which would give casual players something to work towards.