Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Liang	Student ID:	19821986
Other name(s):	Justin Junhao		
Unit name:	Data Structures and Algorithms	Unit ID:	COMP1002
Lecturer / unit coordinator:	Valerie Maxville		
Assessment:	Assignment		

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: JL Date of signature: 17/10/21

(By submitting this form, you indicate that you agree with all the above text.)

Assignment Report

User Guide

Overview of Program

The purpose of the program gameofcatz is to allow a “World” to be represented as a graph and determine that paths that can be taken throughout the “World”. The program would then be able to rank these routes based on the distance or resistance faced. For example, in a snakes and ladder game, the program would be able to find all routes from the start to the end and determine the shortest paths by avoiding all the snakes and utilising the ladders.

Using the Program

Format of Input File

```
Node label code
Edge label label code
Ncode code weight
Ecode code weight
Start label
Target label
```

Simulation Mode

To run Simulation Mode enter: `java gameofcatz -s [inputfile] [outputfile]`

Reads from an input file containing information about the world. Automatically generates and rank the routes, writing them to the output file given on the command line.

Interactive Mode

To run Interactive Mode, enter: `java gameofcatz -i`

```
Interactive Mode
============================
(1) Load input file
(2) Node operations
(3) Edge operations
(4) Parameter tweaks
(5) Display graph
(6) Display world
(7) Change targets
(8) Generate routes
(9) Display routes
(10) Save network
(11) Exit
```

Once in interactive mode, a menu will pop up. This menu allows you to change aspects of the World to your preference. You can either create a World from scratch or import a World from a file and edit that World.

Description of Classes

gameofcatz
Contains the main method to accept command line arguments and determine which mode (Simulation or Interactive) to enter, it is the class used to run the entire program. This class is created due to the specification of the assignment requiring gameofcatz.java

DSAGraph
A data structure that stores nodes that can be connected to each other through an edge. The edge can be seen as a relationship between the two nodes. The class was created so that it could be used to represent the "World".

DSAHashTable
A data structure that allows a key value pair to be stored inside. It converts the key into an index in an array where the value gets stored. It was created as DSAGraph is dependent on it to store its vertices and edges and it is used to store the mappings of the node and edges.

DSAHeap
A tree like data structure that is always partially ordered. It uses DSAHeapEntry object which contains a priority and value pair. The heap is a max heap with the DSAHeapEntry with the highest priority always at the top. It was created to be used in conjunction with DSAGraph to allow the paths generated to be stored and sorted with the use of Heapsort.

DSALinkedList
A dynamic data structure that uses the references of ListNodes to store information in a "chain". It is dynamic and can grow in size. It was created to be used for file reading and storing of targets.

InteractiveMode
Stores static methods used for the Interactive Menu. It includes methods used to enter the various sub-menus and user inputs. It was created the encapsulate Interactive Menu in one class. Some user input methods could have possibly been in the same class, however there were too few of them to warrant an individual class.

SimulationMode
The purpose of this class is to encapsulate methods used to run the simulation of the "World", it processes the data from the file input, generates and rank the routes. Created so that all methods used for the Simulation is in a class together

Operations
Stores all node and edge operations methods used for the Interactive Menu, it stores static methods that are called from the InteractiveMode class. It was created to reduce size of InteractiveMode and improve cohesion between methods.

ModifyTarget
Stores methods used to change the start and target in Interactive Mode. It is created to reduce the number of methods in InteractiveMode and put cohesive methods together.

Display
Store methods used to display the graph and the World in Interactive Mode. Again, created to reduce cluttering in InteractiveMode class.

ParameterTweak
Stores methods used in the Interactive Mode to tweak the mappings of the World. Created to put more cohesive methods together.

FileIO
Stores static methods related to File IO. Includes reading from text and serialized files and writing to text and serialized files. Created to improve readability, as all methods are highly cohesive.

World
Used to store relevant data structures that represents the "World". It contains a graph, which is used to represent the game and Hashtables used to store the mappings for the edge and node codes. It also stores the start and target labels used to generate the routes. This was created to reduce the number of parameters passed into methods and because the "World" could not be represented without one those data structures.

Justification of Decisions:

World
Used a Graph to represent the World as it was required in the assignment specifications.

Underlying structure in Graph
Hashtables were used as an underlying structure in Graphs to store its vertices and edges. It was selected due to its O(1) time complexity for accessing, deleting and inserting. This was important as the assignment performs a large amount of these operations on the node and edges in the graph. This would therefore significantly improve efficiency compared to if LinkedLists were used, as its time complexity for those operations are on average O(N). Hence Hashtables were used in favour of LinkedLists to store the vertices and edges.

Mapping of Node and Edge codes
Hashtables were used store the mapping of codes for Node and Edges. This is because the very nature of mapping a code to value is similar how Hashtables work, which maps a key to a value. Choosing other data structures would require creation of a new Object to store the code and value pair before inserting into its structure. The applications of the Node and Edge maps also requires a lot of accessing, deleting and inserting, therefore Hashtables would be suitable as its time complexity for those operations are O(1). Hence Hashtables were used to store the mapping of codes.

Storing of Targets
Use of LinkedLists to store the target(s). This is due to its dynamic structure and O(N) time complexity when iterating. Since the number of targets cannot be determined easily beforehand, a dynamic structure would be better compared to using an array which is a static structure and

cannot be expanded. The uses for the target(s) usually require all of them to be used at once, therefore O(N) time complexity for iterating of LinkedLists would be more efficient than the O(N + K) time complexity of iterating through a Hashtable. Therefore, LinkedLists were used to store the target(s) over arrays or Hashtables.

Storing of File data
Used LinkedLists to store the data received from a file. Since the amount of data in the file is difficult to determine (Have to go through and read the file once first), it would be more efficient to use a dynamic structure like LinkedList instead of a static one like an array to store the contents of the file.

Finding simple paths
Only found the simple paths in the World as the type of graph that could be used was not specified. If a cyclic graph was used to represent the World, the number of possible paths would be infinite. Therefore, I have only implemented a path finding algorithm that would only search for simple paths.

Storing routes and its cost
Used a Heap to store the routes and its cost. This is because a Heap uses HeapEntries which contains a value and its priority. Therefore, it is perfect in storing the routes and its associated cost. Other data structures would require creating another Object to store the route and it's cost before inserting into its structure. Since the routes needed to be ranked, being in a Heap also allows it to utilise HeapSort which has a sorting time complexity of O(N Log N) making the sorting and ranking of routes fast. Hence a Heap was used to store the routes and cost.

Discussion of Code

Simulation Mode:
Upon receiving the filename from the arguments taken from the command line, it uses the filterFile() method from FileIO class to remove extra information from the file and stores the relevant data in a LinkedList. The LinkedList that contains the file data is iterated twice, the first iteration extracts the mappings for both edges and nodes and then the second iteration grabs the actual nodes and edges. This is done to ensure all mappings are retrieved before the actual nodes and edges in the case where file is not in the right format. Getting the mappings first also allows the nodes and edges that are going to be stored in the graphs to be validated first before it is inserted, in case that there is an unknown mapping tagged to a node or edge.

Once the graph data is stored, it then starts generating all paths between indicated start and target(s). It starts of first by getting the number of paths there is, this is to allow the Heap to be constructed as it is not a dynamic data structure. It then utilises Depth First Search to recursively find all simple paths, appending the current node label to the path String and calculating the cost between two vertices for every recursion. Once it reaches the target node, it stores its current path String and cost into a Heap for use later. My interpretation of the blocks was that it caused a delay, meaning that it still traverses that route, but the cost for the routes would extremely high, not allowing it to be ranked high during the ranking of the routes. I also interpreted the extra targets as alternative finish, meaning that it regenerates the path from start to the next target.

Before writing to the file, it uses Heapsort to rank the routes with the ones with the lowest cost at the top, it then writes all the paths to the file given through the command line.

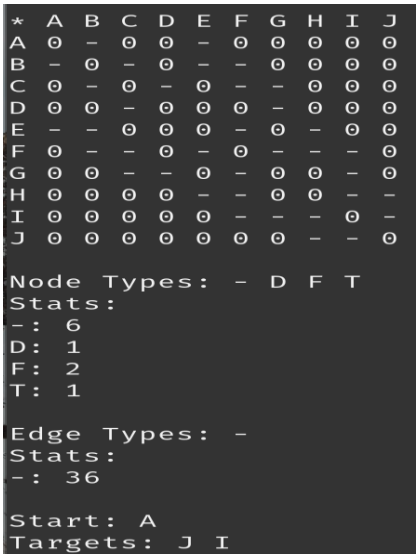
Interactive Mode:
The loading of file input, generating and ranking routes aspects of this mode has already been covered in the simulation mode. The Interactive mode however allows for information of the World to be modified; this was able to be done by allowing the users to directly alter the data structures using user input through its mutator methods.

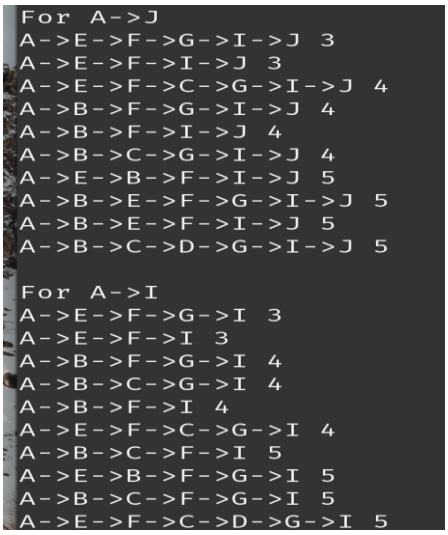
I have implemented a few features that would make the Interactive World more interactive.
The first set of features I implement was allowing the addition of Node and Edge Types, this was relatively easy to implement as it was just a matter of using the put method of a Hashtable to insert the Node or Edge Type of choice. The second feature I have implemented is being able to add and change the start and targets. The changing of targets was a little challenging as it required adding of a remove method in my LinkedList. This would make the world more interactive as now a file input is not required to make the World usable (able to generate routes and display) and can instead be created just through user inputs.

Code Showcase

Scenario 1
Showcasing the feature of adding more targets using the interactive mode and my interpretation of the node values.

1) Read gameofcatz.txt using the File Input Option in the interactive menu.
2) Press 7 to change targets, followed by pressing 2 to add a new target.
3) Enter I as the new target.
4) Display the world by pressing 6. Pressing 2 when prompt to save file.
5) Generating and display the ranked routes by pressing 8 followed by 9.

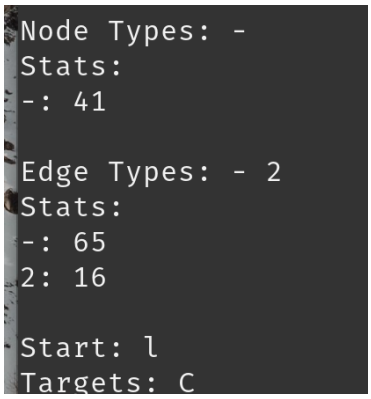


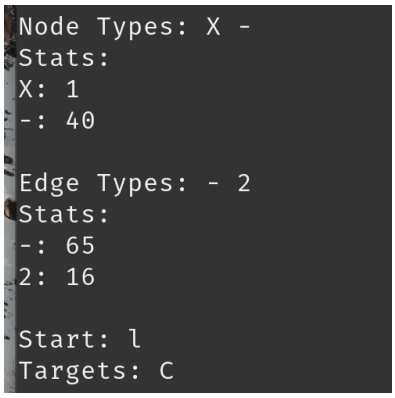


The first ranked route of A→J and A→I have the same weight even though A->J has one more path travelled. This is due to my implementation of treating the node value as a delay. So therefore, it would not consider the node value if it were the last vertex as it does not go past that vertex, hence it does not use the -1 value for I, therefore displaying the same cost.

Scenario 2:
Adding a new node type and updating a node so that it uses the new node type
Displaying old and new world to showcase changes.

1) read amazing2.txt using the file input option in Interactive menu.
2) press 4 to enter parameters menu
3) press 3 to add Node Type
4) Enter X followed by 10
5) Press 5 to exit to main menu
6) Press 2 followed by 4
7) Enter e followed by X





The code output on the left is the original amazing2.txt data, while the one on the right shows the addition of a new node type X into the graph.

Scenario 3:
Creating a small world using only interactive mode.

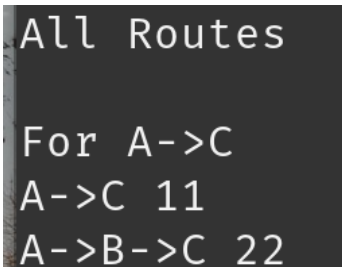
1) Load into Interactive Mode
2) Press 4 followed 3 and then enter X and 1 (Adding node type)
3) Press 4 and then enter – and 10 (Adding edge Type)
4) Press 5 to return to main menu

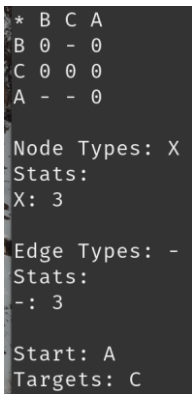
5) Press 2 and then 2 again (Inserting node)
6) Enter A followed by X.

7) Repeat step 5 for B and C.
8) Press 5 to exit to main menu

9) Press 3 followed by 2 (Insert Edge)
10) Enter A followed by B, followed by -
11) Press 2 again, Enter A followed by B and then -
12) Press 2 again, Enter B followed by C and then -
13) Press 2 again, Enter A followed by C and then -

14) Press 5 to exit to main menu (Adding Start and Target)
15) Press 7 followed by 1 then A
16) Press 7 followed by 2 then C
17) Press 8 then followed by 9





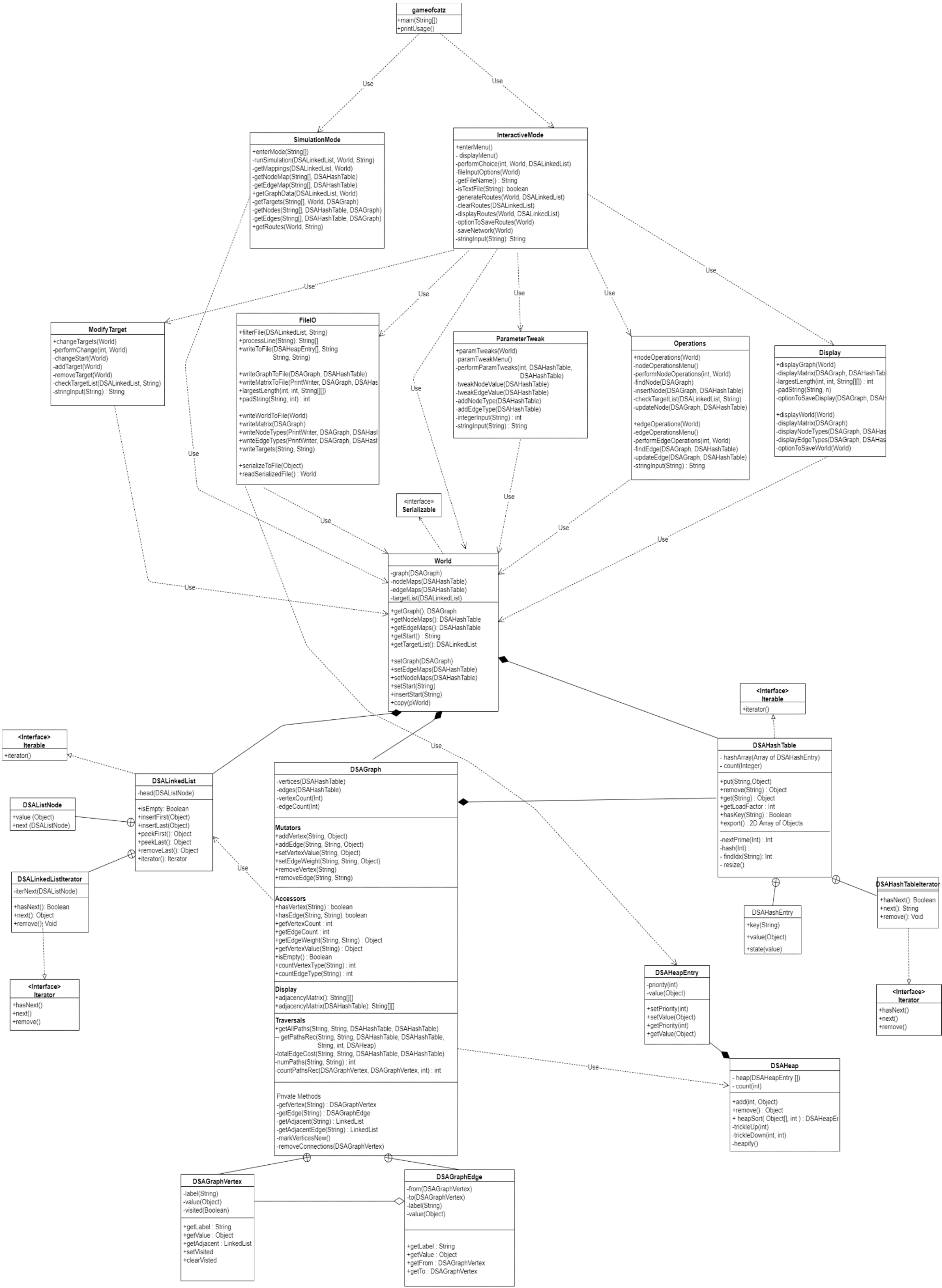
World generated only through interactive mode.

Conclusion

One area my implementation could be improved is the way it handles exceptions. Due to not knowing how to create my own exception class, I have been catching and throwing Java exceptions which would not be good for readability of my code. By creating my own exceptions, it makes the exceptions thrown more related to the error caught.

An extension that I would like to add is providing a graphical representation of the graph to make the graph easier to comprehend and view.

UML DIAGRAM


gameofcatz
+main(String[])
+printUsage()
Use
Use
SimulationMode
+enterMode(String[])
-runSimulation(DSALinkedList, World, String)
-getMappings(DSALinkedList, World)
-getNodeMap(String[], DSAHashTable)
-getEdgeMap(String[], DSAHashTable)
+getGraphData(DSALinkedList, World)
-getTargets(String[], World, DSAGraph)
-getNodes(String[], DSAHashTable, DSAGraph)
-getEdges(String[], DSAHashTable, DSAGraph)
+getRoutes(World, String)
InteractiveMode
+enterMenu()
- displayMenu()
-performChoice(int, World, DSALinkedList)
-fileInputOptions(World)
-getFileName() : String
-isTextFile(String): boolean
-generateRoutes(World, DSALinkedList)
-clearRoutes(DSALinkedList)
-displayRoutes(World, DSALinkedList)
-optionToSaveRoutes(World)
-saveNetwork(World)
-stringInput(String): String
Use
Use
Use
Use
Use
Use
Use
ModifyTarget
+changeTargets(World)
-performChange(int, World)
-changeStart(World)
-addTarget(World)
-removeTarget(World)
-checkTargetList(DSALinkedList, String)
-stringInput(String) : String
FileIO
+filterFile(DSALinkedList, String)
+processLine(String): String[]
+writeToFile(DSAHeapEntry[], String
String, String)
+writeGraphToFile(DSAGraph, DSAHashTable)
+writeMatrixToFile(PrintWriter, DSAGraph, DSAHas
+largestLength(int, int, String[][])
+padString(String, int) : int
+writeWorldToFile(World)
+writeMatrix(DSAGraph)
+writeNodeTypes(PrintWriter, DSAGraph, DSAHasl
+writeEdgeTypes(PrintWriter, DSAGraph, DSAHasl
+writeTargets(String, String)
+serializeToFile(Object)
+readSerializedFile() : World
ParameterTweak
+paramTweaks(World)
-paramTweakMenu()
-performParamTweaks(int, DSAHashTable,
DSAHashTable)
-tweakNodeValue(DSAHashTable)
-tweakEdgeValue(DSAHashTable)
-addNodeType(DSAHashTable)
-addEdgeType(DSAHashTable)
-integerInput(String) : int
-stringInput(String) : String
Operations
+nodeOperations(World)
-nodeOperationsMenu()
-performNodeOperations(int, World)
-findNode(DSAGraph)
-insertNode(DSAGraph, DSAHashTable)
-checkTargetList(DSALinkedList, String)
-updateNode(DSAGraph, DSAHashTable)
+edgeOperations(World)
-edgeOperationsMenu()
-performEdgeOperations(int, World)
-findEdge(DSAGraph, DSAHashTable)
-updateEdge(DSAGraph, DSAHashTable)
-stringInput(String) : String
Display
+displayGraph(World)
-displayMatrix(DSAGraph, DSAHashTab
-largestLength(int, int, String[][]) : int
-padString(String, n)
-optionToSaveDisplay(DSAGraph, DSAH
+displayWorld(World)
-displayMatrix(DSAGraph)
-displayNodeTypes(DSAGraph, DSAHas
-displayEdgeTypes(DSAGraph, DSAHas
-optionToSaveWorld(World)
Use
Use
Use
Use
Use
Use
«interface»
Serializable
World
-graph(DSAGraph)
-nodeMaps(DSAHashTable)
-edgeMaps(DSAHashTable)
-targetList(DSALinkedList)
+getGraph(): DSAGraph
+getNodeMaps(): DSAHashTable
+getEdgeMaps(): DSAHashTable
+getStart() : String
+getTargetList(): DSALinkedList
+setGraph(DSAGraph)
+setEdgeMaps(DSAHashTable)
+setNodeMaps(DSAHashTable)
+setStart(String)
+insertStart(String)
+copy(pWorld)
<Interface>
Iterable
+iterator()
Use
DSAHashTable
- hashArray(Array of DSAHashEntry)
- count(Integer)
+put(String,Object)
+remove(String) : Object
+get(String) : Object
+getLoadFactor : Int
+hasKey(String) : Boolean
+export() : 2D Array of Objects
-nextPrime(Int) : Int
-hash(Int) :
- findIdx(String): Int
- resize()
<Interface>
Iterable
+iterator()
DSALinkedList
-head(DSAListNode)
+isEmpty: Boolean
+insertFirst(Object)
+insertLast(Object)
+peekFirst(): Object
+peekLast(): Object
+removeLast(): Object
+iterator(): Iterator
DSAListNode
+value (Object)
+next (DSAListNode)
DSALinkedListIterator
-iterNext(DSAListNode)
+hasNext(): Boolean
+next(): Object
+remove(): Void
<Interface>
Iterator
+hasNext()
+next()
+remove()
Use
DSAGraph
-vertices(DSAHashTable)
-edges(DSAHashTable)
-vertexCount(Int)
-edgeCount(Int)
Mutators
+addVertex(String, Object)
+addEdge(String, String, Object)
+setVertexValue(String, Object)
+setEdgeWeight(String, String, Object)
+removeVertex(String)
+removeEdge(String, String)
Accessors
+hasVertex(String) : boolean
+hasEdge(String, String): boolean
+getVertexCount : int
+getEdgeCount : int
+getEdgeWeight(String, String) : Object
+getVertexValue(String) : Object
+isEmpty() : Boolean
+countVertexType(String) : int
+countEdgeType(String) : int
Display
+adjacencyMatrix(): String[][]
+adjacencyMatrix(DSAHashTable): String[][]
Traversals
+getAllPaths(String, String, DSAHashTable, DSAHashTable)
-- getPathsRec(String, String, DSAHashTable, DSAHashTable,
String, int, DSAHeap)
-totalEdgeCost(String, String, DSAHashTable, DSAHashTable)
-numPaths(String, String) : int
-countPathsRec(DSAGraphVertex, DSAGraphVertex, int) : int
Private Methods
-getVertex(String) : DSAGraphVertex
-getEdge(String) : DSAGraphEdge
-getAdjacent(String) : LinkedList
-getAdjacentEdge(String) : LinkedList
-markVerticesNew()
-removeConnections(DSAGraphVertex)
DSAHashEntry
+key(String)
+value(Object)
+state(value)
DSAHashTableIterator
+hasNext(): Boolean
+next(): String
+remove(): Void
<Interface>
Iterator
+hasNext()
+next()
+remove()
DSAHeapEntry
-priority(int)
-value(Object)
+setPriority(int)
+setValue(Object)
+getPriority(int)
+getValue(Object)
Use
DSAHeap
- heap(DSAHeapEntry [])
- count(int)
+add(int, Object)
+remove() : Object
+ heapSort(Object[], int) : DSAHeapEr
-trickleUp(int)
-trickleDown(int, int)
-heapify()
DSAGraphVertex
-label(String)
-value(Object)
-visited(Boolean)
+getLabel : String
+getValue : Object
+getAdjacent : LinkedList
+setVisited
+clearVisted
DSAGraphEdge
-from(DSAGraphVertex)
-to(DSAGraphVertex)
-label(String)
-value(Object)
+getLabel : String
+getValue : Object
+getFrom : DSAGraphVertex
+getTo : DSAGraphVertex

Traceability Matrix

1	gameofcatz menu	1.1 Displays Usage when run without arguments	gameofcatz.main()	UnitTestgameofcatz: Script 1.1
		1.2 Displays Interactive Menu when run with -i	gameofcatz.main()	UnitTestgameofcatz: Script 1.2
		1.3 Displays Usage when run with only -s	gameofcatz.main()	UnitTestgameofcatz: Script 1.3
		1.4 Displays Usage when run with only 2 arguments	gameofcatz.main()	UnitTestgameofcatz: Script 1.4
		1.5 Runs SimulationMode when run with correct arguments	gameofcatz.main()	UnitTestgameofcatz: Script 1.5
2	Simulation Mode	2.1 Displays error when given missing file	SimulationMode.enterMode()	UnitTestSimulationMode: Script 2.1
		2.2 Write file to correct output file with ranked routes.	SimulationMode.runSimulation()	UnitTestSimulationMode: Script 2.2
	Interactive Mode			
3	Load files	3.1 Load Text File error checks	InteractiveMode.fileInputOptions()	LoadFile.txt: Script 3.1
		3.2 Display correct graph and world when text file read successful	InteractiveMode.fileInputOptions()	LoadFile.txt: Script 3.2
		3.3 Save and read serialized file works.	InteractiveMode.fileInputOptions()	LoadFile.txt: Script 3.3
4	Node Operations	4.1 Find performs correctly and give appropriate errors	Operations.nodeOperations()	NodeOperations.txt: Script 4.1
		4.2 Insert performs correctly and give appropriate errors.	Operations.nodeOperations()	NodeOperations.txt: Script 4.2
		4.3 Delete performs correctly and give appropriate errors.	Operations.nodeOperations()	NodeOperations.txt: Script 4.3
		4.4 Update performs correctly and give appropriate errors	Operations.nodeOperations()	NodeOperations.txt: Script 4.4
5	Edge Operations	5.1 Find performs correctly and give appropriate errors	Operations.edgeOperations()	EdgeOperations.txt: Script 5.1
		5.2 Add performs correctly and give appropriate errors	Operations.edgeOperations()	EdgeOperations.txt: Script 5.2
		5.3 Remove performs correctly and give appropriate errors	Operations.edgeOperations()	EdgeOperations.txt: Script 5.3
		5.4 Update performs correctly and give appropriate errors	Operations.edgeOperations()	EdgeOperations.txt: Script 5.4
6	Parameter Tweaks	6.1 tweak node type works and give appropriate errors	ParameterTweak.paramTweak()	ParamTweaks.txt: Script 6.1
		6.2 tweak edge type works and give appropriate errors	ParameterTweak.paramTweak()	ParamTweaks.txt: Script 6.2
		6.3 add nodetype works and give appropriate errors	ParameterTweak.paramTweak()	ParamTweaks.txt: Script 6.3
		6.4 add edgeType works and give appropriate errors	ParameterTweak.paramTweak()	ParamTweaks.txt: Script 6.4
7	Display Graph	7.1 Graph displayed is correct	Display.displayGraph()	DisplayGraph.txt: Script 7.1
		7.2 Able to save correct graph display to file.	Display.displayGraph()	DisplayGraph.txt: Script 7.2
8	Display World	8.1 World displayed is correct	Display.displayWorld()	DisplayWorld.txt: Script 8.1
		8.2 Able to save correct world display	Display.displayWorld()	DisplayWorld.txt: Script 8.2
9	Change Targets	9.1 Changing start target works and handles unknown node .	AlterTarget.changeStart()	ChangeTargets.txt:Script 9.1
		9.2 Adding target works and handles unknown nodes.	AlterTarget.addTarget()	ChangeTargets.txt:Script 9.2
		9.3 Removing target works and handles unknown targets.	AlterTarget.removeTarget()	ChangeTargets.txt:Script 9.3
10	Getting routes	10.2 display routes and generate routes is correct	InteractiveMode.displayRoutes()	RoutesTest.txt: Script 10.1
		10.3 write routes to file is correct	InteractiveMode.optionToSaveRoutes()	RoutesTest.txt: Script 10.2
11	Save network	10.2 Check that save network is correct.	InteractiveMode.saveNetwork()	Checked in Script 3.3

References

Reuse of code used in Practicals 03, 06, 07 and 08

LinkedList TestHarness by Connor Beardsmore

Used algorithm to find all paths from source to destination.
https://www.geeksforgeeks.org/find-paths-given-source-destination/