MAJOR PROJECT REPORT

GEOM2159 – Geospatial Programming

GitHub Link

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.
When you're ready to add your content, just click here and start typing.]

Bradley Denney
s3661151

Contents

Introduction 2
Literature Review 2
Planning 3
Steps 3
Timeline 3
Functional Units 4
Construct Full Pseudocode 4
Description 4
Reflection 4
Run Process in ArcGIS & QGIS 4
Description 4
Results 5
Create Base Script 5
Test/Break/Improve Base Script 6
Description 6
Move to QGIS Processing Tool 6
Process 7
Test/Break/Improve QGIS Processing Tool 8
Description 8
Reflection 8
Migrate to Other Devices 9
Conclusion 9
Limitations 9
References 10
Appendix 1 - Pseudocode 11
Breakdown 11
Initialise 11
Road Buffer 11
Hydro Buffer 11
Merge Buffer 11
Initial Buffer 11
Iterative Section 11
Clean-up 12
QGIS Functions to be Used 12
Appendix 2 – Base Script 13
Appendix 3 – Improved Script 18
Appendix 4 – Processing Tool 22
Appendix 4 – Improved Tool 30

Major Project Report

Introduction

This project aims to create a tool that can calculate the area of land (minus roads and rivers) that can be covered with certain volumes of waste products from a broiler farm.

This is a tool which, when run, will determine and visualise the area that can be covered by an amount of concentrated waste, accounting for roads and creeks (which needs no fertilising). This will demonstrate to clients the area that can be covered if they are smart about their waste and use it as valuable soil-enriching nutrients. The constructed buffer can show clearly the area that can be covered, and account for different land uses around the area. This tool will not, however, account for slope variation. This may be an extension for this tool in the future.

Currently, I do this task on ArcGIS Pro with the help of an Excel spreadsheet. This is prone to errors with me copying the wrong number from the spreadsheet, or errors in my formulae, and so lacks the Quality Assurance and Quality Control (QA/QC) I desire in other processes of my work. The whole thing must be done manually due to certain steps requiring values from other items created in the process, and so a straightforward task can take ages, and any adjustment means the whole process must be done over again. This wastes my time and the client's money.

The intended user for this process is me! Or any other GIS professional who needs to do this task. However, the intention is to create a tool that is portable and self-explanatory enough that anyone with a basic knowledge of geography and planning can use the tool (provided adequate knowledge of QGIS). This tool will greatly simplify the task, reduce the time to create a visualisation of this phenomenon, and allow for more rigorous quality assurance through reliable, repeatable results. The product will be compared to a manually created result to ensure correct results.

Literature Review

Chicken farms, while profitable, produce many by-products that are typically either disposed of or sold offshore to the agriculture sector. Turnell et al. (2007) summarised ways the Australian farming sector has become more efficient in the usage of this waste. The paper mentioned using the waste from these broiler farms to work towards creating circular economies. However, the study lacked numerical analysis of the area that may be covered for certain concentrations of waste.

Mkhabela (2004) used mathematical models to predict these areas and determine areas that may be covered by broiler fertiliser at a variety of concentrations. These values and findings of this report are valuable, but the paper lacked visual representation of these findings. While further studies, such as Memon et al. (2016), conclude that broiler waste is a good fertiliser for certain crops, there continues to be a lack of proposal of a model to visualise this benefit. As such, there is also no mention of accounting for land use or transportation in these area calculations.

Planning

Steps

Below is a breakdown of the steps to be taken throughout the project:

1. Construct full pseudocode
2. Run process manually using ArcGIS & QGIS
3. Create a Base Script
    1. Initialise script
    2. Create a road buffer
    3. Create a hydro buffer
    4. Merge & Dissolve buffers
    5. Create initial waste buffer
    6. Iterate:
        - Clip by network buffer
        - Use the area of clip to create a new buffer
    7. Report the change in the area covered
    8. Save new shapefile/delete temp files
4. Test/break/improve Base Script
5. Move Base Script to QGIS Processing tool
6. Test/break/improve Processing tool
7. Test Processing tool on other devices, other scenarios and ensure portability

Timeline

A proposed timeline was suggested and can be seen in Table 1. Some steps have taken less time than budgeted, but most steps have been more complicated than previously expected.

Table 1. A proposed timeline for tasks

Step	Plan	Time Estimated (hours)	Time Taken (hours)
Pseudocode	Done (I misread the proposal rubric, so I've already done this part. See Appendix 1)	2.5	2.5
ArcGIS & QGIS	Run process on ArcGIS Pro using a method I know works. Then replicate this on QGIS to see all steps that are taken and how they could be improved/simplified.	2	5
Base script	Use QGIS Python console and script to construct a working base-script that will work for a specific scenario. (Big one)	9	5
Test / Improve	Use the constructed script in different scenarios to see if I can break it, then improve those shortcomings.	2	5
Processing Tool	Migrate base script to processing tool by making it more generalised and user-friendly.	5	10
Test / Improve	Test the processing tool to find shortcomings and fix them.	2	1
Portability	Take the tool and try it on other devices (such as RMIT computers or a Mac). Fix up any interoperability issues.	2	N/A
Total		24.5	28.5

Functional Units

Construct Full Pseudocode

Time Allocated: 2.5 hours – Time Taken: 2.5

Description

This step involved determining the necessary tasks to be completed in each sub-section of mining the Base Script. The breakdown of proposed steps, and QGIS functions to be used in each can be seen in Appendix 1.

Reflection

While this was a useful step for determining the steps that needed to be taken, I was kind of guessing most processes. I think this step would have been better done after performing the task on ArcGIS Pro or QGIS. This process did, however, refamiliarise me with the overall task, and provide a framework for the later creation of the Base Script. In hindsight, I believe this would have been better created after a period of heavy manual usage, so I could identify aspects of the process that could be improved through automation (not just human error and time).

Run Process in ArcGIS & QGIS

Time Allocated: 2 hours – Time Taken: 5 hours

Description

Running the process manually allows for a better understanding of the tasks required in each step of the process. There was only 1 minor difference in terminology between the two software. The individual steps taken can be seen below:

1. Set up project space
2. Define central point
3. Add hydro data
4. Add road data
5. Create 50 metre Hydro buffer
6. Create 40 metre Road buffer
7. Remove Hydro and Road data
8. Merge Hydro and Road buffer feature classes (MergeBuffer)
9. Delete Hydro and Road buffers
10. Merge Hydro and Road buffer features (DissolveBuffer)
11. Delete MergeBuffer
12. Determine volume of Nitrogen
13. Determine concentration of Nitrogen required
14. Calculate area of Buffer0
15. Calculate radius of Buffer0
16. Create initial Buffer0
17. Clip DissolveBuffer by Buffer0 (Clip1)
18. Determine area of Clip1
19. Calculate area for Buffer1 (Area of Buffer0 + Area of Clip1)
20. Calculate radius for Buffer1
21. Create Buffer1
22. Clip DissolveBuffer by Buffer1 (Clip2)
23. Determine area of Clip 2
24. Calculate area of Buffer2 (Area of Buffer1 + (Area of Clip2 - Area of Clip1))
25. Delete Buffer2 and Clip1
26. Calculate radius of Buffer 2

27. Create Buffer2
28. Clip DissolveBuffer by Buffer2 (Clip3)
29. Determine area of Clip 3
30. Calculate area of Buffer3 (Area of Buffer2 + (Area of Clip3 - Area of Clip2))
31. Delete Buffer2 and Clip2
32. Calculate radius of Buffer 3
33. Create Buffer3
34. Delete DissolveBuffer
35. Calculate area increase (Area of Buffer3 – Area of Buffer0)
36. Calculate percent increase (((Area of Buffer3 / Area of Buffer0) – 1) * 100)
37. Delete Buffer0

Results

A screenshot of the result of the two software can be seen in Figure 1, and a summary of the results of running the process can be seen in Table 2. Further testing confirms most of this difference in area results from the segmentation of arc-length in QGIS.

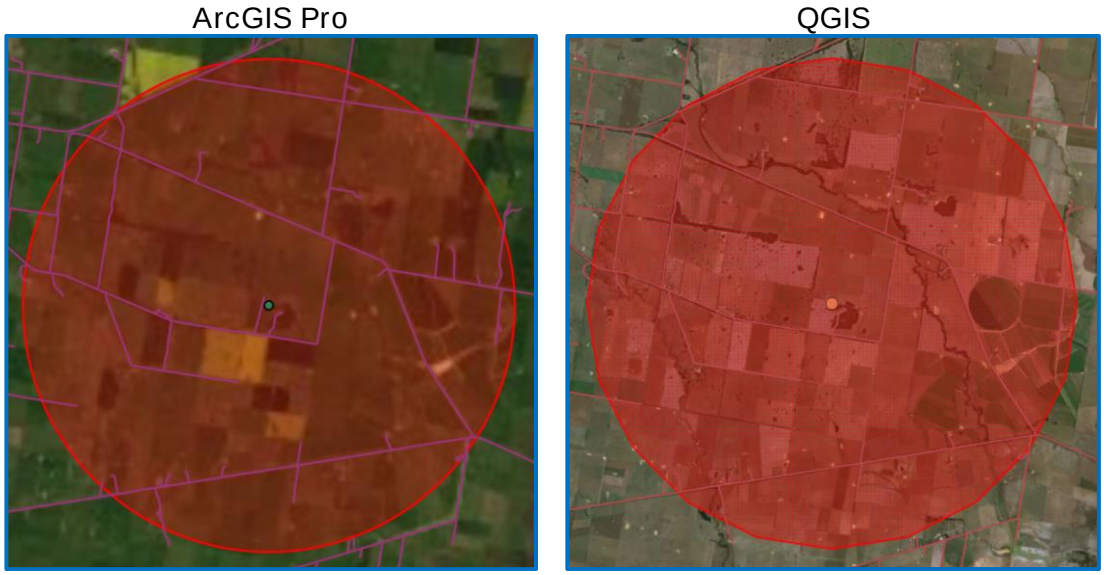

Figure 1. Resultant outputs from ArcGIS Pro and QGIS

Table 2. ArcGIS Pro vs QGIS manual accuracy

	ArcGIS Pro	QGIS
areaIncrease (Ha)	899	874
percentIncrease (%)	21	20

Create Base Script

Time Allocated: 9 hours – Time Taken: 5 hours

This set was the ‘guts’ of the project. It involved creating the Base Script which all further progress is based on (see Appendix 2). While this was not the final copy of the script used, it provided a Python script that, when run, would provide a resultant buffer and information about the process post-running. The results can be seen in Table 3.

Table 3. Base Script accuracy

	Base Script
areaIncrease (Ha)	832
percentIncrease (%)	19

Test/Break/Improve Base Script

Time Allocated: 2 hours – Time Taken: 5 hours

Description

In this step, I worked to improve the Base Script. This involved making general changes to the script to make it more streamlined, manageable, and concise. The resultant code can be seen in Appendix 3. The major changes made to the Base Script are detailed below:

1. Move the process to a function with specified parameters

This allowed the process to be more easily tested under different inputs. Additionally, it allowed it the transition to Processing Tool to be more easily implemented.

2. Allow for different compounds to be calculated

Allowing the user to input different compounds allows this tool to finally be used for all of Nitrogen, Phosphorus, and Potassium, as per the client's request.

3. Put repeat buffer in while loop

Pushing the repetitive process into a while loop not only made the while script more manageable but also made it easier to make any necessary changes throughout the script further. Only 1 change had to be made when previously it was 3 changes. Finally, this change also allowed for a theoretically unlimited number of iterations.

4. Increase segments of area buffers

This was only a small change, and arbitrary in the change made to the number. Each quarter section of a circle was now broken into 10 segments instead of the default 5. This change increased the area covered, to one comparable to the process done manually on ArcGIS Pro.

Table 4. Improved Script accuracy

	Improved Script
areaIncrease (Ha)	892
percentIncrease (%)	21

Move to QGIS Processing Tool

Time Allocated: 5 hours – Time Taken: 10 hours

This step involved moving the Improved Script to a QGIS Processing Tool. This was by far the most challenging step of the entire project and presented the most errors that could not be easily solved. The new script for the Processing Tool can be found in Appendix 4.

The advantages of moving this process to a Processing Tool are numerous. Such benefits include:

1. Portability

Moving to a tool means the user specifies their own file paths. As such, this tool can theoretically be downloaded by anyone and used on their local system, with no changes made to the script itself.

2. Usability

The QGIS tool interface is much easier and straightforward to be used than the Python console. This means anyone with basic experience of using a GIS will be able to navigate and use the process. Much easier to be used and explained.

3. Alterability

This process can now be done under different conditions (different compounds, broiler mass) with ease. This achieves the goal of enabling the process to be down in minimal time with minute changes. In theory, I could even use this process in a meeting with a client, when previously I would need hours.

Process

This step was huge but can be roughly divided into 3 main steps:

1. Re-doing Module 3 Exercise

This allowed me to see how the boiler template for processing tool works. This was valuable for establishing a base understanding of the processing tool and how it works, however, there were some limitations to the similarities, but beneficial, nonetheless.

2. Filling in Boiler Template

Here, I worked along with the example exercise and worked to start introducing my script to the processing tool template. This was relatively straightforward, and most issues could be solved after some searching for documentation and examples. I self-learned a lot. I used the parameters for the tool seen in Table 5.

Table 5. Processing Tool parameters

Parameter	QgsProcessingParameter type
`INPUT: Input layer`	`QgsProcessingParameterFeatureSource`
`HYDRO: Hydro data layer`	`QgsProcessingParameterFeatureSource`
`ROAD: Road data layer`	`QgsProcessingParameterFeatureSource`
`MASS: Mass of broiler waste`	`QgsProcessingParameterNumber`
`COMPOUND: Compound`	`QgsProcessingParameterEnum`
`ITERATIONS: Iterations`	`QgsProcessingParameterNumber`
`OUTPUT: Output buffer layer`	`QgsProcessingParameterFeatureSink`

3. Troubleshooting (feat. Dr Griffin)

THIS STEP WAS A MASSIVE PAIN, NOT GONNA LIE. This was when I ran into too many brick walls and needed to call in the Big Brains. Together with Amy, I worked through each error as it came up. This was pretty damn tedious, but beneficial in that I found additional beneficial resources. I still learnt a lot in this section; however, I lack the deep understanding I had previously over the entire process. TLDR; I know what I did, but I don't exactly know how I did it and I'm not sure I could replicate the process again in a different setting. Figure 2 shows the resultant user interface of the tool with some parameters manually filled.

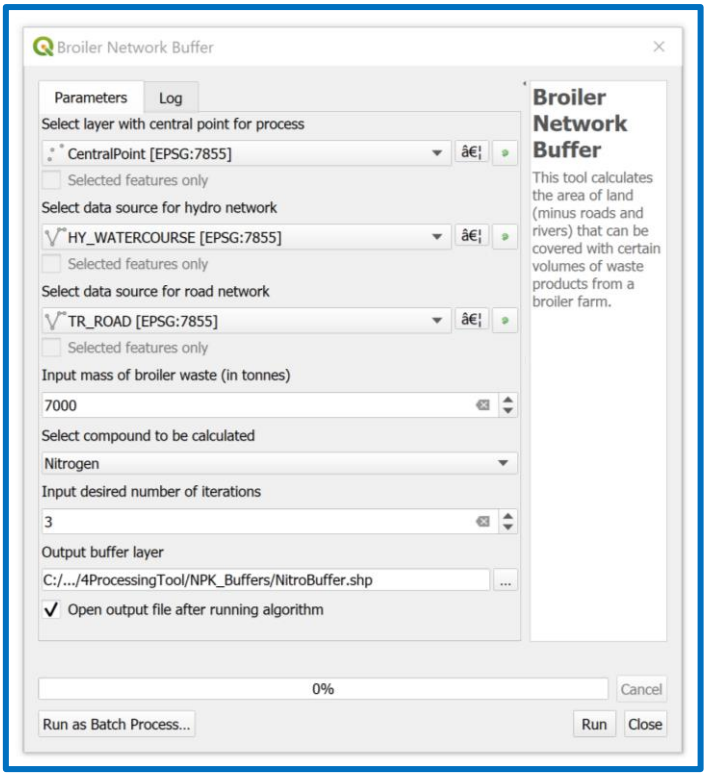

Figure 2. Processing Tool user interface

Test/Break/Improve QGIS Processing Tool

Time Allocated: 2 hours – Time Taken: 1

Description

This step involved making minor edits to the code of the processing tool to made it work better. These made the program a little less susceptible to faults and will make it easier to edit and improve in the future. The script was also commented to enhance readability and understanding.

Reflection

This was only done partially. Given more time, I would have liked to further improve the Processing Tool. Such improvements would include:

1. Implementing constraints on the parameters (e.g. no negative values for number iterations)
2. Provide default values for parameters
3. Increase usability of the user interface (more guide and explanations)
4. Increasing efficiency of code (somehow reduce the repetition, perhaps abstraction?)

Migrate to Other Devices

Time Allocated: 2 hours – Time Taken: N/A

Unfortunately, this step could not be completed due to running out of time.

However, the way this code was implemented and constructed resulted in a tool that is not dependant on any local values (e.g. file paths). Further, I have confirmation that loading the tool will not corrupt your QGIS application. Thus, I can theoretically conclude that the product will work on a variety of machines, including WindowsOS and MacOS.

Conclusion

This study has detailed the development of the creation of a tool to calculate and visualise area that may be covered by fertiliser from a broiler farm. The aim of creating a QGIS processing tool has been achieved, and the tool can be used reliably for commercial applications, provided the limitations of the area calculations are accounted for.

Limitations

This tool accounts for the lack of need for fertiliser on roadways and near rivers. It does not account for slope (runoff), land use (urban vs rural), or accessibility. These may be areas for further development to enhance the commercial application areas of a tool such as the one created.

Furthermore, with more time, the processing tool created may be improved and advanced. This is not due to a small project timeframe, but rather poor time management. With better time management, this tool may be recreated and improved upon.

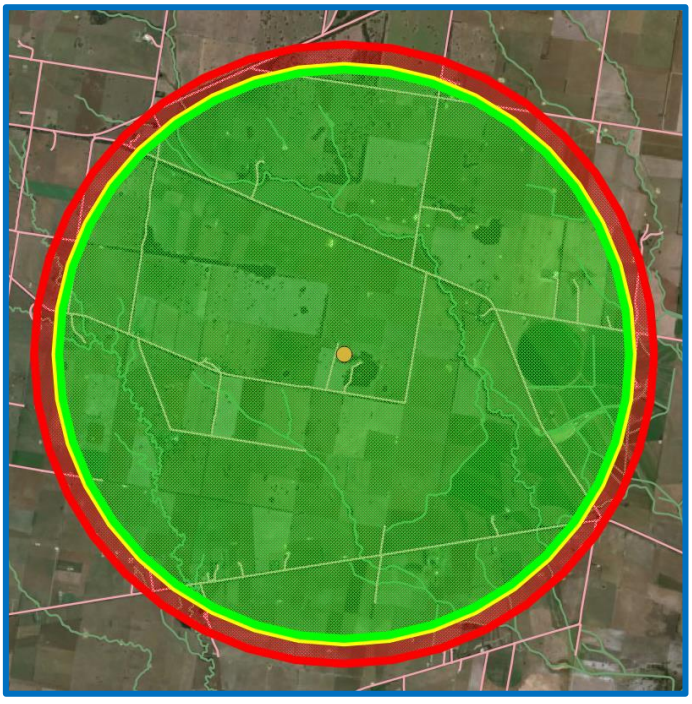

Figure 3. Final NPK buffers

References

Memon, I., Kumbhar, M. and Noonari, S., 2016. Economics of Poultry Waste Use as a Fertilizer in Sindh Pakistan. Journal of Fisheries & Livestock Production, 4(2), pp.1-9.

Mkhabela, T.S., 2004. Substitution of Fertiliser with Poultry Manure: is this Economically Viable? Agrekon, 43(3), pp.347-356.

Turnell, J.R., Faulkner, R.D. and Hinch, G.N., 2007. Recent Advances in Australian Broiler Litter Utilisation. World's Poultry Science Journal, 63(2), pp.223-231.

Appendix 1 - Pseudocode

Breakdown

Initialise

Set file path of data folder

Set road, hydro files

Set concentration, waste, centre point

Add files to data frame (`.addVectorLayer()`)

Road Buffer

Define road buffer parameters

Run road buffer (`.processing('native:buffer')`)

Add buffer to data frame

Hydro Buffer

Define hydro buffer parameters

Run hydro buffer (`.processing('native:buffer')`)

Add buffer to data frame (`.addVectorLayer()`)

Merge Buffer

Merge layers into one layer (`.processing('qgis:mergevectorlayers')`)

Add merged layer to data frame (`.addVectorLayer()`)

Dissolve features together (`.processing('qgis:dissolve')`)

Initial Buffer

Calculate area, and therefore, radius to be covered

Define initial buffer parameters

Run initial buffer (`.processing('native:buffer')`)

Add buffer to data frame (`.addVectorLayer()`)

Iterative Section

Clip

Clip road buffer by initial buffer (`.processing('qgis:clip')`)

Return area of clip

Recalculate new area, therefore radius

Buffer

Define repeat buffer parameters

Run repeat buffer (`.processing('native:buffer')`)

Add buffer to data frame (`.addVectorLayer()`)

Clean-up

Print area of initial buffer

Print area of new buffer

Save buffer as shapefile (**`.writeAsVectorFormat()`**)

Delete temp files (**`deleteShapeFile()`**)

QGIS Functions to be Used

Function	Parameters

`QgisInterface.addVectorLayer()`	`vectorLayerPath : QString`
	`baseName : QString`
	`providerKey : QString`

`processing.run('native:buffer')`	`INPUT : QgsRasterLayer`
	`DISTANCE : int`
	`OUTPUT : str`

`processing.run('native:mergevectorlayers')`	`LAYERS : list[QgsMapLayer]`
	`CRS : QgsProcessingParameterCrs`
	`OUTPUT : str`

`processing.run('native:dissolve')`	`INPUT : QgsVectorLayer`
	`FIELD : str`
	`OUTPUT : str`

`processing.run('native:clip')`	`INPUT : QgsVectorLayer`
	`OVERLAY : QgsVectorLayer`
	`OUTPUT : str`

`QgsVectorFileWriter.writeAsVectorFormat()`	`layer : QgsVectorLayer`
	`filename : QString`
	`destCRS : QgsCoordinateReferenceSystem`
	`filetype : QString`

`QgsVectorFileWriter.deleteShapeFile()`	`filename : QString`

Appendix 2 – Base Script

Code

```
# Set file path to data folder
filePath = 'C:\\RMIT\\Geospatial Programming\\MajorProject\\3BaseScript\\'
# Set file for Central Point
pointFile = 'Data\\CentralPoint.shp'
# Set file for Hydro data
hydroFile = 'Data\\HY_WATERCOURSE.shp'
# Set file for Roads data
roadFile = 'Data\\TR_ROAD.shp'
# Set mass of Nitrogen
massNitro = 215000
# Set concentration of Nitrogen
concNitro = 0.005
# Add Hydro layer to data frame
hydroLayer = iface.addVectorLayer(f'{filePath}{hydroFile}', 'Hydro', 'ogr')
# Add CentralPoint layer to data frame
roadLayer = iface.addVectorLayer(f'{filePath}{roadFile}', 'Roads', 'ogr')
# Add CentralPoint layer to data frame
pointLayer = iface.addVectorLayer(f'{filePath}{pointFile}', 'Central
Point', 'ogr')

# Define parameters for Hydro buffer
hydroBuffParameters = {
'INPUT':hydroLayer,
'DISTANCE':50,
'DISSOLVE':True,
'OUTPUT':f'{filePath}Temp\\hydroBufferFile.shp'
}
# Run Hydro buffer process
processing.run('native:buffer',hydroBuffParameters)
# Add buffer layer to data frame
hydroBuffer =
iface.addVectorLayer(f'{filePath}Temp\\hydroBufferFile.shp','HydroBuffer',’
ogr')

# Define parameters for Road buffer
roadBuffParameters = {
'INPUT':roadLayer,
'DISTANCE':40,
'DISSOLVE':True,
'OUTPUT':f'{filePath}Temp\\roadBufferFile.shp'
}
# Run Road buffer process
processing.run('native:buffer',roadBuffParameters)
# Add buffer layer to data frame
roadBuffer =
iface.addVectorLayer(f'{filePath}Temp\\roadBufferFile.shp','RoadBuffer','og
r')

# Define parameters for Merge process
mergeParameters = {
'LAYERS':[hydroBuffer,roadBuffer],
'OUTPUT':f'{filePath}Temp\\mergeFile.shp'
}
# Run Merge process
processing.run('qgis:mergevectorlayers',mergeParameters)
# Add merged layer to data frame
mergeBuffer =
iface.addVectorLayer(f'{filePath}Temp\\mergeFile.shp','MergeBuffer','ogr')
```

```
# Define parameters for Dissolve process
dissolveParameters = {
'INPUT':mergeBuffer,
'OUTPUT':f'{filePath}Temp\\dissolveFile.shp'
}
# Run Dissolve process
processing.run('qgis:dissolve',dissolveParameters)
# Add merged layer to data frame
dissolveBuffer =
iface.addVectorLayer(f'{filePath}Temp\\dissolveFile.shp','Dissolve
Buffer','ogr')

# Calculate area of Buffer0
areaBuffer0 = massNitro / concNitro
# Calculate distance of Buffer0
distanceBuffer0 = math.sqrt(areaBuffer0/math.pi)

# Define parameters for Buffer0
parametersBuffer0 = {
'INPUT':pointLayer,
'DISTANCE':distanceBuffer0,
'OUTPUT':f'{filePath}Temp\\Buffer0File.shp'
}
# Run Buffer0 process
processing.run('native:buffer',parametersBuffer0)
# Add Buffer0 layer to data frame
Buffer0 =
iface.addVectorLayer(f'{filePath}Temp\\Buffer0File.shp','Buffer0','ogr')

# Create list of Buffer0 features
featuresBuffer0 = Buffer0.getFeatures()
# Iterate through features
for feature in featuresBuffer0:
    # Determine feature area
    areaBuffer0 = feature.geometry().area()

# Define parameters for Clip1
parametersClip1 = {
'INPUT':dissolveBuffer,
'OVERLAY':Buffer0,
'OUTPUT':f'{filePath}Temp\\Clip1File.shp'
}
# Run Clip1 process
processing.run('qgis:clip',parametersClip1)
# Add Clip1 layer to data frame
Clip1 =
iface.addVectorLayer(f'{filePath}Temp\\Clip1File.shp','Clip1','ogr')

# Create list of Clip1 features
featuresClip1 = Clip1.getFeatures()
# Iterate through features
for feature in featuresClip1:
    # Determine feature area
    areaClip1 = feature.geometry().area()

# Calculate Buffer 1 area
areaBuffer1 = areaBuffer0 + areaClip1
# Calculate Buffer 1 distance
distanceBuffer1 = math.sqrt(areaBuffer1/math.pi)
```

```
# Define parameters for Buffer1
parametersBuffer1 = {
'INPUT':pointLayer,
'DISTANCE':distanceBuffer1,
'OUTPUT':f'{filePath}Temp\\Buffer1File.shp'
}
# Run Buffer1 process
processing.run('native:buffer',parametersBuffer1)
# Add Buffer1 layer to data frame
Buffer1 =
iface.addVectorLayer(f'{filePath}Temp\\Buffer1File.shp','Buffer1','ogr')

# Create list of Buffer1 features
featuresBuffer1 = Buffer1.getFeatures()
# Iterate through features
for feature in featuresBuffer1:
    # Determine feature area
    areaBuffer1 = feature.geometry().area()

# Define parameters for Clip2
parametersClip2 = {
'INPUT':dissolveBuffer,
'OVERLAY':Buffer1,
'OUTPUT':f'{filePath}Temp\\Clip2File.shp'
}
# Run Clip2 process
processing.run('qgis:clip',parametersClip2)
# Add Clip2 layer to data frame
Clip2 =
iface.addVectorLayer(f'{filePath}Temp\\Clip2File.shp','Clip2','ogr')

# Create list of Clip2 features
featuresClip2 = Clip2.getFeatures()
# Iterate through features
for feature in featuresClip2:
    # Determine feature area
    areaClip2 = feature.geometry().area()

# Calculate Buffer 2 area
areaBuffer2 = areaBuffer1 + (areaClip2 - areaClip1)
# Calculate Buffer 2 distance
distanceBuffer2 = math.sqrt(areaBuffer2/math.pi)

# Define parameters for Buffer2
parametersBuffer2 = {
'INPUT':pointLayer,
'DISTANCE':distanceBuffer2,
'OUTPUT':f'{filePath}Temp\\Buffer2File.shp'
}
# Run Buffer2 process
processing.run('native:buffer',parametersBuffer2)
# Add Buffer2 layer to data frame
Buffer2 =
face.addVectorLayer(f'{filePath}Temp\\Buffer2File.shp','Buffer2','ogr')

# Create list of Buffer2 features
featuresBuffer2 = Buffer2.getFeatures()
# Iterate through features
for feature in featuresBuffer2:
    # Determine feature area
    areaBuffer2 = feature.geometry().area()
```

```
# Define parameters for Clip3
parametersClip3 = {
'INPUT':dissolveBuffer,
'OVERLAY':Buffer2,
'OUTPUT':f'{filePath}Temp\\Clip3File.shp'
}
# Run Clip3 process
processing.run('qgis:clip',parametersClip3)
# Add Clip3 layer to data frame
Clip3 =
iface.addVectorLayer(f'{filePath}Temp\\Clip3File.shp','Clip3','ogr')

# Create list of Clip3 features
featuresClip3 = Clip3.getFeatures()
# Iterate through features
for feature in featuresClip3:
    # Determine feature area
    areaClip3 = feature.geometry().area()

# Calculate Buffer 3 area
areaBuffer3 = areaBuffer2 + (areaClip3 - areaClip2)
# Calculate Buffer 2 distance
distanceBuffer3 = math.sqrt(areaBuffer3/math.pi)

# Define parameters for Buffer3
parametersBuffer3 = {
'INPUT':pointLayer,
'DISTANCE':distanceBuffer3,
'OUTPUT':f'{filePath}Buffer3.shp'
}
# Run Buffer3 process
processing.run('native:buffer',parametersBuffer3)
# Add Buffer3 layer to data frame
Buffer3 = iface.addVectorLayer(f'{filePath}Buffer3.shp','Buffer3','ogr')

# Calculate area increase
areaIncrease = areaBuffer3 - areaBuffer0
# Calculate percent increase
pcIncrease = ((areaBuffer3/areaBuffer0)-1)*100

# Remove Temporary map layers
QgsProject.instance().removeMapLayer(hydroLayer)
QgsProject.instance().removeMapLayer(roadLayer)
QgsProject.instance().removeMapLayer(hydroBuffer)
QgsProject.instance().removeMapLayer(roadBuffer)
QgsProject.instance().removeMapLayer(mergeBuffer)
QgsProject.instance().removeMapLayer(dissolveBuffer)
QgsProject.instance().removeMapLayer(Buffer0)
QgsProject.instance().removeMapLayer(Clip1)
QgsProject.instance().removeMapLayer(Buffer1)
QgsProject.instance().removeMapLayer(Clip2)
QgsProject.instance().removeMapLayer(Buffer2)
QgsProject.instance().removeMapLayer(Clip3)

# Remove most Temp files
for files in os.listdir(f'{filePath}\\Temp'):
    QgsVectorFileWriter.deleteShapeFile(f'{filePath}\\Temp\\{files}')

# Print areaBuffer3
print(f'Area of new buffer is {int(round(areaBuffer3/10000,0))} Ha')
```

```
# Print areaIncrease
print(f'Process increases area covered by {int(round(areaIncrease/10000))}
Ha')
# Print pcIncrease
print(f'This is {round(pcIncrease,0)}% bigger than original buffer')
```

Appendix 3 – Improved Script

Code

```
def broilerBuffer(compound, massBroilerWaste, iterations):
    # Set mass & concentration of compound
    if compound == 'Nitrogen':
        massCompound = massBroilerWaste * 30.714286
        concCompound = 0.005
    elif compound == 'Phosphorus':
        massCompound = massBroilerWaste * 14.142857
        concCompound = 0.0027
    elif compound == 'Potassium':
        massCompound = massBroilerWaste * 13.428571
        concCompound = 0.0025

    # Set file path to data folder
    filePath =
'C:\\RMIT\\GeospatialProgramming\\MajorProject\\3BaseScript\\'
    # Set file for Central Point
    pointFile = 'Data\\CentralPoint.shp'
    # Set file for Hydro data
    hydroFile = 'Data\\HY_WATERCOURSE.shp'
    # Set file for Roads data
    roadFile = 'Data\\TR_ROAD.shp'
    # Add Hydro layer to data frame
    hydroLayer = iface.addVectorLayer(f'{filePath}{hydroFile}', 'Hydro',
'ogr')
    # Add CentralPoint layer to data frame
    roadLayer = iface.addVectorLayer(f'{filePath}{roadFile}', 'Roads',
'ogr')
    # Add CentralPoint layer to data frame
    pointLayer = iface.addVectorLayer(f'{filePath}{pointFile}', 'Central
Point', 'ogr')

    # Delete most Temp files
    for files in os.listdir(f'{filePath}\\Temp'):
        QgsVectorFileWriter.deleteShapeFile(f'{filePath}\\Temp\\{files}')

    # Define parameters for Hydro buffer
    hydroBuffParameters = {
    'INPUT' : hydroLayer,
    'DISTANCE' : 50,
    'DISSOLVE' : True,
    'OUTPUT' : f'{filePath}Temp\\hydroBufferFile.shp'
    }
    # Run Hydro buffer process
    processing.run('native:buffer', hydroBuffParameters)
    # Add buffer layer to data frame
    hydroBuffer =
iface.addVectorLayer(f'{filePath}Temp\\hydroBufferFile.shp', 'Hydro
Buffer', 'ogr')

    # Define parameters for Road buffer
    roadBuffParameters = {
    'INPUT' : roadLayer,
    'DISTANCE' : 40,
    'DISSOLVE' : True,
    'OUTPUT' : f'{filePath}Temp\\roadBufferFile.shp'
    }
    # Run Road buffer process
    processing.run('native:buffer', roadBuffParameters)
```

```
    # Add buffer layer to data frame
    roadBuffer =
iface.addVectorLayer(f'{filePath}Temp\\roadBufferFile.shp', 'Road Buffer',
'ogr')

    # Define parameters for Merge process
    mergeParameters = {
    'LAYERS' : [hydroBuffer, roadBuffer],
    'OUTPUT' : f'{filePath}Temp\\mergeFile.shp'
    }
    # Run Merge process
    processing.run('qgis:mergevectorlayers', mergeParameters)
    # Add merged layer to data frame
    mergeBuffer = iface.addVectorLayer(f'{filePath}Temp\\mergeFile.shp',
'Merge Buffer', 'ogr')

    # Define parameters for Dissolve process
    dissolveParameters = {
    'INPUT' : mergeBuffer,
    'OUTPUT' : f'{filePath}Temp\\dissolveFile.shp'
    }
    # Run Dissolve process
    processing.run('qgis:dissolve', dissolveParameters)
    # Add merged layer to data frame
    dissolveBuffer =
iface.addVectorLayer(f'{filePath}Temp\\dissolveFile.shp', 'Dissolve
Buffer', 'ogr')

    # Establish reference lists
    listBuff = [0]
    listClip = [0]
    listAreaBuff = [0]
    listAreaClip = [0]
    listDistBuff = [0]

    # Calculate area of Buffer0
    listAreaBuff[0] = massCompound / concCompound
    # Calculate distance of Buffer0
    listDistBuff[0] = math.sqrt(listAreaBuff[0] / math.pi)

    # Define parameters for Buffer0
    parametersBuffer = {
    'INPUT' : pointLayer,
    'DISTANCE' : listDistBuff[0],
    'SEGMENTS' : 10,
    'OUTPUT' : f'{filePath}Temp\\Buffer0File.shp'
    }
    # Run Buffer0 process
    processing.run('native:buffer', parametersBuffer)
    # Add Buffer0 layer to data frame
    listBuff[0] = iface.addVectorLayer(f'{filePath}Temp\\Buffer0File.shp',
'Buffer0', 'ogr')

    for count in range (1, iterations + 1):
        # Define parameters for Clip
        parametersClip = {
        'INPUT' : dissolveBuffer,
        'OVERLAY' : listBuff[count - 1],
        'OUTPUT' : f'{filePath}Temp\\Clip{count}File.shp'
        }
        # Run Clip process
```

```
        processing.run('qgis:clip', parametersClip)
        # Add Clip layer to data frame

listClip.append(iface.addVectorLayer(f'{filePath}Temp\\Clip{count}File.shp'
, f'Clip{count}', 'ogr'))

        # Create list of Clip features
        featuresClip = listClip[count].getFeatures()
        # Iterate through features
        for feature in featuresClip:
            # Determine feature area
            listAreaClip.append(feature.geometry().area())

        # Calculate Buffer area
        listAreaBuff.append(listAreaBuff[count - 1] + (listAreaClip[count]
- listAreaClip[count - 1]))
        # Calculate Buffer distance
        listDistBuff.append(math.sqrt(listAreaBuff[count] / math.pi))

        # Cause final buffer to be saved permanently
        if count == iterations:
            # Define parameters for Buffer
            parametersBuffer = {
            'INPUT' : pointLayer,
            'DISTANCE' : listDistBuff[count],
            'SEGMENTS' : 10,
            'OUTPUT' : f'{filePath}WasteBuffer.shp'
            }
            # Run Buffer process
            processing.run('native:buffer', parametersBuffer)
            # Add Buffer layer to data frame

listBuff.append(iface.addVectorLayer(f'{filePath}WasteBuffer.shp',
f'Buffer{count}' ,'ogr'))
        else:
            # Define parameters for Buffer
            parametersBuffer = {
            'INPUT' : pointLayer,
            'DISTANCE' : listDistBuff[count],
            'SEGMENTS' : 10,
            'OUTPUT' : f'{filePath}Temp\\Buffer{count}File.shp'
            }
            # Run Buffer process
            processing.run('native:buffer', parametersBuffer)
            # Add Buffer layer to data frame

listBuff.append(iface.addVectorLayer(f'{filePath}Temp\\Buffer{count}File.sh
p', f'Buffer{count}', 'ogr'))

    # Calculate area increase
    areaIncrease = listAreaBuff[iterations] - listAreaBuff[0]
    # Calculate percent increase
    pcIncrease = ((listAreaBuff[iterations] / listAreaBuff[0]) - 1) * 100

    # Remove Temporary map layers
    QgsProject.instance().removeMapLayer(hydroLayer)
    QgsProject.instance().removeMapLayer(roadLayer)
    QgsProject.instance().removeMapLayer(hydroBuffer)
    QgsProject.instance().removeMapLayer(roadBuffer)
    QgsProject.instance().removeMapLayer(mergeBuffer)
    QgsProject.instance().removeMapLayer(dissolveBuffer)
```

```
    for count in range(1, iterations+1):
        QgsProject.instance().removeMapLayer(listBuff[count - 1])
        QgsProject.instance().removeMapLayer(listClip[count])

    # Print areaBuffer3
    print(f'{massBroilerWaste}t of broiler waste contains
{int(round(massCompound / 1000))}t of {compound}, which covers
{int(round(listAreaBuff[iterations] / 10000))} Ha')
    # Print areaIncrease
    print(f'Process increases area covered by {int(round(areaIncrease /
10000))} Ha')
    # Print pcIncrease
    print(f'This is {round(pcIncrease)}% larger than original area')


# Run function
broilerBuffer('Nitrogen', 7000, 3)
```

Appendix 4 – Processing Tool

Code

```
# -*- coding: utf-8 -*-

"""
***************************************************************************
*                                                                         *
*   This program is free software; you can redistribute it and/or modify  *
*   it under the terms of the GNU General Public License as published by  *
*   the Free Software Foundation; either version 2 of the License, or     *
*   (at your option) any later version.                                   *
*                                                                         *
***************************************************************************
"""

from qgis.PyQt.QtCore import QCoreApplication
from qgis.core import (QgsProcessing,
                       QgsFeature,
                       QgsFeatureSink,
                       QgsProcessingException,
                       QgsProcessingAlgorithm,
                       QgsProcessingParameterFeatureSource,
                       QgsProcessingParameterFeatureSink,
                       QgsProcessingParameterNumber,
                       QgsProcessingParameterEnum,
                       QgsProcessingFeatureSource)
import os
import math
from qgis.utils import iface
from qgis import processing


class BroilerNetworkBuffer(QgsProcessingAlgorithm):
    """
    This is an example algorithm that takes a vector layer and
    creates a new identical one.

    It is meant to be used as an example of how to create your own
    algorithms and explain methods and variables used to do it. An
    algorithm like this will be available in all elements, and there
    is not need for additional work.

    All Processing algorithms should extend the QgsProcessingAlgorithm
    class.
    """

    # Constants used to refer to parameters and outputs. They will be
    # used when calling the algorithm from another algorithm, or when
    # calling from the QGIS console.

    INPUT = 'INPUT'
    HYDRO = 'HYDRO'
    ROAD = 'ROAD'
    MASS = 'MASS'
    COMPOUND = 'COMPOUND'
```

```
    ITERATIONS = 'ITERATIONS'
    OUTPUT = 'OUTPUT'

    def tr(self, string):
        """
        Returns a translatable string with the self.tr() function.
        """
        return QCoreApplication.translate('Processing', string)

    def createInstance(self):
        return BroilerNetworkBuffer()

    def name(self):
        """
        Returns the algorithm name, used for identifying the algorithm. This
        string should be fixed for the algorithm, and must not be localised.
        The name should be unique within each provider. Names should contain
        lowercase alphanumeric characters only and no spaces or other
        formatting characters.
        """
        return 'broilernetworkbuffer'

    def displayName(self):
        """
        Returns the translated algorithm name, which should be used for any
        user-visible display of the algorithm name.
        """
        return self.tr('Broiler Network Buffer')

    def group(self):
        """
        Returns the name of the group this algorithm belongs to. This string
        should be localised.
        """
        return self.tr('Example scripts')

    def groupId(self):
        """
        Returns the unique ID of the group this algorithm belongs to. This
        string should be fixed for the algorithm, and must not be localised.
        The group id should be unique within each provider. Group id should
        contain lowercase alphanumeric characters only and no spaces or other
        formatting characters.
        """
        return 'examplescripts'

    def shortHelpString(self):
        """
        Returns a localised short helper string for the algorithm. This
string
        should provide a basic description about what the algorithm does and
the
        parameters and outputs associated with it..
        """
        return self.tr("This tool calculates the area of land (minus roads
and rivers) that can be covered with certain volumes of waste products from
a broiler farm.")
```

```
    def initAlgorithm(self, config=None):
        """
        Here we define the inputs and output of the algorithm, along
        with some other properties.
        """

        # We add the input vector features source. It can have any kind of
        # geometry.
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.INPUT,
                self.tr('Select layer with central point for process'),
                [QgsProcessing.TypeVectorPoint]
            )
        )

        # We add the input vector features source. It can have any kind of
        # geometry.
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.HYDRO,
                self.tr('Select data source for hydro network'),
                [QgsProcessing.TypeVectorLine]
            )
        )

        # We add the input vector features source. It can have any kind of
        # geometry.
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.ROAD,
                self.tr('Select data source for road network'),
                [QgsProcessing.TypeVectorLine]
            )
        )

        # We add the input vector features source. It can have any kind of
        # geometry.
        self.addParameter(
            QgsProcessingParameterNumber(
                self.MASS,
                self.tr('Input mass of broiler waste (in tonnes)'),
            )
        )

        # We add the input vector features source. It can have any kind of
        # geometry.
        self.addParameter(
            QgsProcessingParameterEnum(
                self.COMPOUND,
                self.tr('Select compound to be calculated'),
                ['Nitrogen','Phosphorus','Potassium']
            )
        )

        # We add the input vector features source. It can have any kind of
```

```
        # geometry.
        self.addParameter(
            QgsProcessingParameterNumber(
                self.ITERATIONS,
                self.tr('Input desired number of iterations'),
            )
        )

        # We add a feature sink in which to store our processed features
(this
        # usually takes the form of a newly created vector layer when the
        # algorithm is run in QGIS).
        self.addParameter(
            QgsProcessingParameterFeatureSink(
                self.OUTPUT,
                self.tr('Output buffer layer')
            )
        )

    def processAlgorithm(self, parameters, context, feedback):
        """
        Here is where the processing itself takes place.
        """

        # Retrieve the feature source and sink. The 'dest_id' variable is
used
        # to uniquely identify the feature sink, and must be included in the
        # dictionary returned by the processAlgorithm function.
        pointFile = self.parameterAsSource(
            parameters,
            self.INPUT,
            context
        )
        hydroFile = self.parameterAsSource(
            parameters,
            self.HYDRO,
            context
        )
        roadFile = self.parameterAsSource(
            parameters,
            self.ROAD,
            context
        )
        massBroilerWaste = self.parameterAsDouble(
            parameters,
            self.MASS,
            context
        )
        compound = self.parameterAsEnum(
            parameters,
            self.COMPOUND,
            context
        )
        iterations = self.parameterAsInt(
            parameters,
            self.ITERATIONS,
            context
```

```
        )

        # If source was not found, throw an exception to indicate that the
algorithm
        # encountered a fatal error. The exception text can be any string,
but in this
        # case we use the pre-built invalidSourceError method to return a
standard
        # helper text for when a source cannot be evaluated
        if pointFile is None:
            raise QgsProcessingException(self.invalidSourceError(parameters,
self.INPUT))

        (sink, dest_id) = self.parameterAsSink(
            parameters,
            self.OUTPUT,
            context,
            pointFile.fields(),
            3,
            pointFile.sourceCrs()
        )

        # Send some information to the user
        feedback.pushInfo('CRS                                              is
{}'.format(pointFile.sourceCrs().authid()))

        # Set mass & concentration of compound
        if parameters[self.COMPOUND] == 0:
            massCompound = parameters[self.MASS] * 30.714286
            concCompound = 0.005
        elif parameters[self.COMPOUND] == 1:
            massCompound = parameters[self.MASS] * 14.142857
            concCompound = 0.0027
        elif parameters[self.COMPOUND] == 2:
            massCompound = parameters[self.MASS] * 13.428571
            concCompound = 0.0025

        # Define parameters for Hydro buffer
        hydroBuffParameters = {
        'INPUT' : parameters[self.HYDRO],
        'DISTANCE' : 50,
        'DISSOLVE' : True,
        'OUTPUT' : 'memory:'
        }
        # Run Hydro buffer process
        hydroBuffer = processing.run('native:buffer', hydroBuffParameters)
        # Add buffer layer to data frame
        #hydroBuffer                                                        =
iface.addVectorLayer(f'{filePath}Temp\\hydroBufferFile.shp', 'Hydro Buffer',
'ogr')

        # Define parameters for Road buffer
        roadBuffParameters = {
        'INPUT' : parameters[self.ROAD],
        'DISTANCE' : 40,
        'DISSOLVE' : True,
        'OUTPUT' : 'memory:'
```

```
        }
        # Run Road buffer process
        roadBuffer = processing.run('native:buffer', roadBuffParameters)
        # Add buffer layer to data frame
        #roadBuffer                                                              =
iface.addVectorLayer(f'{filePath}Temp\\roadBufferFile.shp',  'Road  Buffer',
'ogr')

        # Define parameters for Merge process
        mergeParameters = {
        'LAYERS' : [hydroBuffer["OUTPUT"], roadBuffer["OUTPUT"]],
        'OUTPUT' : 'memory:'
        }
        # Run Merge process
        mergeBuffer       =       processing.run('qgis:mergevectorlayers',
mergeParameters)
        # Add merged layer to data frame
        #mergeBuffer                                                            =
iface.addVectorLayer(f'{filePath}Temp\\mergeFile.shp',    'Merge    Buffer',
'ogr')

        # Define parameters for Dissolve process
        dissolveParameters = {
        'INPUT' : mergeBuffer["OUTPUT"],
        'OUTPUT' : 'memory:'
        }
        # Run Dissolve process
        dissolveBuffer = processing.run('qgis:dissolve', dissolveParameters)
        # Add merged layer to data frame
        #dissolveBuffer                                                         =
iface.addVectorLayer(f'{filePath}Temp\\dissolveFile.shp', 'Dissolve Buffer',
'ogr')

        # Establish reference lists
        listBuff = [0]
        listClip = [0]
        listAreaBuff = [0]
        listAreaClip = [0]
        listDistBuff = [0]

        # Calculate area of Buffer0
        listAreaBuff[0] = massCompound / concCompound
        # Calculate distance of Buffer0
        listDistBuff[0] = math.sqrt(listAreaBuff[0] / math.pi)

        # Define parameters for Buffer0
        parametersBuffer = {
        'INPUT' : parameters[self.INPUT],
        'DISTANCE' : listDistBuff[0],
        'SEGMENTS' : 10,
        'OUTPUT' : 'memory:'
        }
        # Run Buffer0 process
        listBuff[0] = processing.run('native:buffer', parametersBuffer)
        # Add Buffer0 layer to data frame
        #listBuff[0]                                                            =
iface.addVectorLayer(f'{filePath}Temp\\Buffer0File.shp', 'Buffer0', 'ogr')
```

```
        for count in range (1, iterations + 1):
            # Define parameters for Clip
            parametersClip = {
            'INPUT' : dissolveBuffer["OUTPUT"],
            'OVERLAY' : listBuff[count - 1]["OUTPUT"],
            'OUTPUT' : 'memory:'
            }
            # Run Clip process
            listClip.append(processing.run('qgis:clip', parametersClip))
            # Add Clip layer to data frame

#listClip.append(iface.addVectorLayer(f'{filePath}Temp\\Clip{count}File.shp
', f'Clip{count}', 'ogr'))

            # Create list of Clip features
            featuresClip = listClip[count]["OUTPUT"].getFeatures()
            # Iterate through features
            for feature in featuresClip:
                # Determine feature area
                listAreaClip.append(feature.geometry().area())

            # Calculate Buffer area
            listAreaBuff.append(listAreaBuff[count       -       1]       +
(listAreaClip[count] - listAreaClip[count - 1]))
            # Calculate Buffer distance
            listDistBuff.append(math.sqrt(listAreaBuff[count] / math.pi))

            # Cause final buffer to be saved permanently
            if count == iterations:
                # Define parameters for Buffer
                parametersBuffer = {
                'INPUT' : parameters[self.INPUT],
                'DISTANCE' : listDistBuff[count],
                'SEGMENTS' : 10,
                'OUTPUT' : 'memory:'
                }
                # Run Buffer process
                listBuff.append(processing.run('native:buffer',
parametersBuffer))
                # Add Buffer layer to data frame

#listBuff.append(iface.addVectorLayer(f'{filePath}{compound}Buffer.shp',
f'Buffer{count}' ,'ogr'))
            else:
                # Define parameters for Buffer
                parametersBuffer = {
                'INPUT' : parameters[self.INPUT],
                'DISTANCE' : listDistBuff[count],
                'SEGMENTS' : 10,
                'OUTPUT' : 'memory:'
                }
                # Run Buffer process
                listBuff.append(processing.run('native:buffer',
parametersBuffer))
                # Add Buffer layer to data frame
```

```
#listBuff.append(iface.addVectorLayer(f'{filePath}Temp\\Buffer{count}File.s
hp', f'Buffer{count}', 'ogr'))

        # Read the dissolved layer and create output features
        for feature in listBuff[iterations]["OUTPUT"].getFeatures():
            new_feature =  QgsFeature()
            # Set geometry to dissolved geometry
            new_feature.setGeometry(feature.geometry())
            # Set attributes from sum_unique_values dictionary that we had
computed
            new_feature.setAttributes(["Id", 0])
            sink.addFeature(new_feature, QgsFeatureSink.FastInsert)

        # Calculate area increase
        areaIncrease = listAreaBuff[iterations] - listAreaBuff[0]
        # Calculate percent increase
        pcIncrease = ((listAreaBuff[iterations] / listAreaBuff[0]) - 1) *
100

        # Print areaBuffer3
        feedback.pushInfo(f'{parameters[self.MASS]}t of broiler waste
contains {int(round(massCompound / 1000))}t of fertiliser, which covers
{int(round(listAreaBuff[iterations] / 10000))} Ha')
        # Print areaIncrease
        feedback.pushInfo(f'Process increases area covered by
{int(round(areaIncrease / 10000))} Ha')
        # Print pcIncrease
        feedback.pushInfo(f'This is {round(pcIncrease)}% larger than
original area')

        return {self.OUTPUT: dest_id}
```

Appendix 4 – Improved Tool

Code

```
# -*- coding: utf-8 -*-

"""
This is a tool which, when run, will determine and visualise the area that
can
be covered by an amount of concentrated waste, accounting for roads and
creeks
(which need no fertilising).
This will demonstrate to clients the area that can be covered if they are
smart
about their waste and use it as valuable soil enriching nutrients.
The constructed buffer can show clearly the area that can be covered, and
account for different land uses around the area.  This tool will not, however,
account for slope variation.  This may be an extension for this tool in the
future.
"""

# Import relevant Python and PyQGIS libraries
import math
import os
from qgis import processing
from qgis.core import (QgsFeature,
                       QgsFeatureSink,
                       QgsProcessing,
                       QgsProcessingAlgorithm,
                       QgsProcessingException,
                       QgsProcessingFeatureSource,
                       QgsProcessingParameterEnum,
                       QgsProcessingParameterFeatureSink,
                       QgsProcessingParameterFeatureSource,
                       QgsProcessingParameterNumber)
from qgis.PyQt.QtCore import QCoreApplication
from qgis.utils import iface


# Establish the processing algorithm
class BroilerNetworkBuffer(QgsProcessingAlgorithm):
    """
    This is a tool which, when run, will determine and visualise the area
that
    can be covered by an amount of concentrated waste, accounting for roads
and
    creeks (which need no fertilising).
    This will demonstrate to clients the area that can be covered if they
are
    smart about their waste and use it as valuable soil enriching nutrients.
    The constructed buffer can show clearly the area that can be covered,
and
    account for different land uses around the area.  This tool will not,
    however, account for slope variation.  This may be an extension for this
    tool in the future.
    """
```

```
    # Constants used to refer to parameters and outputs. They will be
    # used when calling the algorithm from another algorithm, or when
    # calling from the QGIS console.

    INPUT = 'INPUT'
    HYDRO = 'HYDRO'
    ROAD = 'ROAD'
    MASS = 'MASS'
    COMPOUND = 'COMPOUND'
    ITERATIONS = 'ITERATIONS'
    OUTPUT = 'OUTPUT'

    def tr(self, string):
        """
        Returns a translatable string with the self.tr() function.
        """
        return QCoreApplication.translate('Processing', string)

    def createInstance(self):
        return BroilerNetworkBuffer()

    def name(self):
        """
        Returns the algorithm name, used for identifying the algorithm.
        """
        return 'broilernetworkbuffer'

    def displayName(self):
        """
        Returns the translated algorithm name.
        """
        return self.tr('Broiler Network Buffer')

    def group(self):
        """
        Returns the name of the group this algorithm belongs to.
        """
        return self.tr('Example scripts')

    def groupId(self):
        """
        Returns the unique ID of the group this algorithm belongs to.
        """
        return 'examplescripts'

    def shortHelpString(self):
        """
        Returns a localised short helper string for the algorithm.
        """
        return self.tr("This tool calculates the area of land (minus roads
and rivers) that can be covered with certain volumes of waste products from
a broiler farm.")

    def initAlgorithm(self, config=None):
        """
        Here we define the inputs and output of the algorithm, along
        with some other properties.
```

```
        """

        # We add the input vector features source. It must be a point layer.
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.INPUT,
                self.tr('Select layer with central point for process'),
                [QgsProcessing.TypeVectorPoint]
            )
        )

        # We add the hydrology data source. It must be a linear network.
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.HYDRO,
                self.tr('Select data source for hydro network'),
                [QgsProcessing.TypeVectorLine]
            )
        )

        # We add the transport data source. It must be a linear network.
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.ROAD,
                self.tr('Select data source for road network'),
                [QgsProcessing.TypeVectorLine]
            )
        )

        # We specify the mass of broiler waste available at the central waste.
        self.addParameter(
            QgsProcessingParameterNumber(
                self.MASS,
                self.tr('Input mass of broiler waste (in tonnes)'),
            )
        )

        # We specify the compound being calculated.
        self.addParameter(
            QgsProcessingParameterEnum(
                self.COMPOUND,
                self.tr('Select compound to be calculated'),
                ['Nitrogen','Phosphorus','Potassium']
            )
        )

        # We specify how many iterations we want to run of this process.
        self.addParameter(
            QgsProcessingParameterNumber(
                self.ITERATIONS,
                self.tr('Input desired number of iterations'),
            )
        )

        # We add a feature sink in which to store our processed feature.
        self.addParameter(
            QgsProcessingParameterFeatureSink(
```

```
                self.OUTPUT,
                self.tr('Output buffer layer')
            )
        )

    def processAlgorithm(self, parameters, context, feedback):
        """
        Here is where the processing itself takes place.
        """

        # Retrieve the feature sources and other parameter values.
        pointFile = self.parameterAsSource(
            parameters,
            self.INPUT,
            context
        )
        hydroFile = self.parameterAsSource(
            parameters,
            self.HYDRO,
            context
        )
        roadFile = self.parameterAsSource(
            parameters,
            self.ROAD,
            context
        )
        massBroilerWaste = self.parameterAsDouble(
            parameters,
            self.MASS,
            context
        )
        compound = self.parameterAsEnum(
            parameters,
            self.COMPOUND,
            context
        )
        iterations = self.parameterAsInt(
            parameters,
            self.ITERATIONS,
            context
        )

        # If source was not found, throw an exception to indicate that the
algorithm encountered a fatal error.
        if pointFile is None:
            raise QgsProcessingException(self.invalidSourceError(parameters,
self.INPUT))

        # Specify information about the output layer.
        (sink, dest_id) = self.parameterAsSink(
            parameters,
            self.OUTPUT,
            context,
            pointFile.fields(),
            3,
            pointFile.sourceCrs()
        )
```

```
        # Set mass & concentration of compound
        if parameters[self.COMPOUND] == 0:
            massCompound = parameters[self.MASS] * 30.714286
            concCompound = 0.005
        elif parameters[self.COMPOUND] == 1:
            massCompound = parameters[self.MASS] * 14.142857
            concCompound = 0.0027
        elif parameters[self.COMPOUND] == 2:
            massCompound = parameters[self.MASS] * 13.428571
            concCompound = 0.0025

        # Define parameters for Hydro buffer
        hydroBuffParameters = {
        'INPUT' : parameters[self.HYDRO],
        'DISTANCE' : 50,
        'DISSOLVE' : True,
        'OUTPUT' : 'memory:'
        }
        # Run Hydro buffer process
        hydroBuffer = processing.run('native:buffer', hydroBuffParameters)

        # Define parameters for Road buffer
        roadBuffParameters = {
        'INPUT' : parameters[self.ROAD],
        'DISTANCE' : 40,
        'DISSOLVE' : True,
        'OUTPUT' : 'memory:'
        }
        # Run Road buffer process
        roadBuffer = processing.run('native:buffer', roadBuffParameters)

        # Define parameters for Merge process
        mergeParameters = {
        'LAYERS' : [hydroBuffer["OUTPUT"], roadBuffer["OUTPUT"]],
        'OUTPUT' : 'memory:'
        }
        # Run Merge process
        mergeBuffer       =       processing.run('qgis:mergevectorlayers',
mergeParameters)

        # Define parameters for Dissolve process
        dissolveParameters = {
        'INPUT' : mergeBuffer["OUTPUT"],
        'OUTPUT' : 'memory:'
        }
        # Run Dissolve process
        dissolveBuffer = processing.run('qgis:dissolve', dissolveParameters)

        # Establish reference lists for loop
        listBuff = [0]
        listClip = [0]
        listAreaBuff = [0]
        listAreaClip = [0]
        listDistBuff = [0]

        # Calculate area of Buffer0
```

```
        listAreaBuff[0] = massCompound / concCompound
        # Calculate distance of Buffer0
        listDistBuff[0] = math.sqrt(listAreaBuff[0] / math.pi)

        # Define parameters for Buffer0
        parametersBuffer = {
        'INPUT' : parameters[self.INPUT],
        'DISTANCE' : listDistBuff[0],
        'SEGMENTS' : 10,
        'OUTPUT' : 'memory:'
        }
        # Run Buffer0 process
        listBuff[0] = processing.run('native:buffer', parametersBuffer)

        # This step runs iterations of the buffer process and clip.
        # It calculates the area of the networks covered by the buffer and
adds it to the waste buffer.
        # This is an iterative process.  More iterations get closer to the
'true' value.
        for count in range (1, iterations + 1):
            # Define parameters for Clip
            parametersClip = {
            'INPUT' : dissolveBuffer["OUTPUT"],
            'OVERLAY' : listBuff[count - 1]["OUTPUT"],
            'OUTPUT' : 'memory:'
            }
            # Run Clip process
            listClip.append(processing.run('qgis:clip', parametersClip))

            # Create list of Clip features
            featuresClip = listClip[count]["OUTPUT"].getFeatures()
            # Iterate through features
            for feature in featuresClip:
                # Determine feature area covered by clipped network buffer
                listAreaClip.append(feature.geometry().area())

            # Calculate Buffer area
            listAreaBuff.append(listAreaBuff[count       -       1]       +
(listAreaClip[count] - listAreaClip[count - 1]))
            # Calculate Buffer distance
            listDistBuff.append(math.sqrt(listAreaBuff[count] / math.pi))

            # Define parameters for Buffer
            parametersBuffer = {
            'INPUT' : parameters[self.INPUT],
            'DISTANCE' : listDistBuff[count],
            'SEGMENTS' : 10,
            'OUTPUT' : 'memory:'
            }
            # Run Buffer process
            listBuff.append(processing.run('native:buffer',
parametersBuffer))

        # Read the Buffer layer and create output features
        for feature in listBuff[iterations]["OUTPUT"].getFeatures():
            new_feature =  QgsFeature()
            # Set geometry to Buffer geometry
```

```
            new_feature.setGeometry(feature.geometry())
            # Set Id so feature can be indexed in Shapefile
            new_feature.setAttributes(["Id", 0])
            sink.addFeature(new_feature, QgsFeatureSink.FastInsert)

        # Calculate area increase
        areaIncrease = listAreaBuff[iterations] - listAreaBuff[0]
        # Calculate percent increase
        pcIncrease = ((listAreaBuff[iterations] / listAreaBuff[0]) - 1) *
100

        # Print area of final Buffer
        feedback.pushInfo(f'{parameters[self.MASS]}t of broiler waste
contains {int(round(massCompound / 1000))}t of fertiliser, which covers
{int(round(listAreaBuff[iterations] / 10000))} Ha')
        # Print area that has been added through this process
        feedback.pushInfo(f'Process increases area covered by
{int(round(areaIncrease / 10000))} Ha')
        # Print percent increase process has provided
        feedback.pushInfo(f'This is {round(pcIncrease)}% larger than
original area')

        # Return final Buffer as ouput layer
        return {self.OUTPUT: dest_id}
```