```
from qgis.core import QgsProcessing
from qgis.core import QgsProcessingAlgorithm
from qgis.core import QgsProcessingMultiStepFeedback
from qgis.core import QgsProcessingParameterVectorLayer
from qgis.core import QgsProcessingParameterRasterDestination
import processing


class Viewshed(QgsProcessingAlgorithm):

    def initAlgorithm(self, config=None):
        self.addParameter(QgsProcessingParameterVectorLayer('newsite', 'New Site',
types=[QgsProcessing.TypeVector], defaultValue=None))
        self.addParameter(QgsProcessingParameterRasterDestination('Viewshed', 'viewshed',
createByDefault=True, defaultValue=None))

    def processAlgorithm(self, parameters, context, model_feedback):
        # Use a multi-step feedback, so that individual child algorithm progress reports are adjusted for
the
        # overall progress through the model
        feedback = QgsProcessingMultiStepFeedback(4, model_feedback)
        results = {}
        outputs = {}

        # Regular points
        alg_params = {
            'CRS': QgsCoordinateReferenceSystem('EPSG:27700'),
            'EXTENT': parameters['newsite'],
            'INSET': 10,
            'IS_SPACING': True,
            'RANDOMIZE': False,
            'SPACING': 25,
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        outputs['RegularPoints'] = processing.run('qgis:regularpoints', alg_params, context=context,
feedback=feedback, is_child_algorithm=True)

        feedback.setCurrentStep(1)
        if feedback.isCanceled():
            return {}

        # Extract by location
        alg_params = {
            'INPUT': outputs['RegularPoints']['OUTPUT'],
            'INTERSECT': parameters['newsite'],
            'PREDICATE': [6],
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        outputs['ExtractByLocation'] = processing.run('native:extractbylocation', alg_params,
context=context, feedback=feedback, is_child_algorithm=True)
```

```
        feedback.setCurrentStep(2)
        if feedback.isCanceled():
            return {}

        # Create viewpoints
        alg_params = {
            'DEM': 'C:/Users/Ben/Documents/2019 Ben/Master GIS UK/Final
Project/Layers/UK/Terrain.tif',
            'OBSERVER_ID': '',
            'OBSERVER_POINTS': outputs['ExtractByLocation']['OUTPUT'],
            'OBS_HEIGHT': 2.3,
            'OBS_HEIGHT_FIELD': '',
            'RADIUS': 3000,
            'RADIUS_FIELD': '',
            'TARGET_HEIGHT': 1.6,
            'TARGET_HEIGHT_FIELD': '',
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        outputs['CreateViewpoints'] = processing.run('visibility:create_viewpoints', alg_params,
context=context, feedback=feedback, is_child_algorithm=True)

        feedback.setCurrentStep(3)
        if feedback.isCanceled():
            return {}

        # Viewshed
        alg_params = {
            'DEM': 'C:/Users/Ben/Documents/2019 Ben/Master GIS UK/Final
Project/Layers/UK/Terrain.tif',
            'OBSERVER_POINTS': outputs['CreateViewpoints']['OUTPUT'],
            'REFRACTION': 0.13,
            'USE_CURVATURE': False,
            'OUTPUT': parameters['Viewshed']
        }
        outputs['Viewshed'] = processing.run('visibility:Viewshed', alg_params, context=context,
feedback=feedback, is_child_algorithm=True)
        results['Viewshed'] = outputs['Viewshed']['OUTPUT']
        return results

    def name(self):
        return 'Viewshed'

    def displayName(self):
        return 'Viewshed'

    def group(self):
        return 'Observer Height'

    def groupId(self):
        return 'Observer Height'
```

```
def createInstance(self):
    return Viewshed()
```