HSC SDD
Major Project Portfolio

Part 3: Final Major Project Submission

JAMES DENOVAN

Kinross Wolarai School

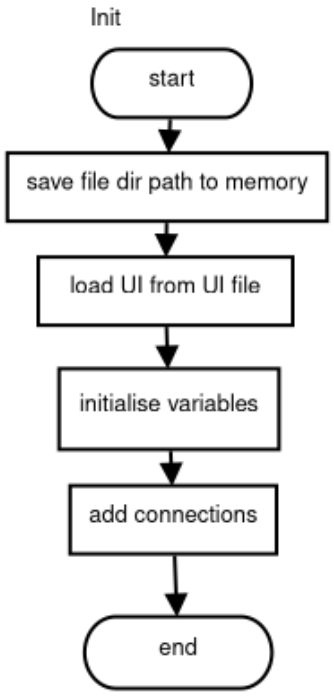
Contents

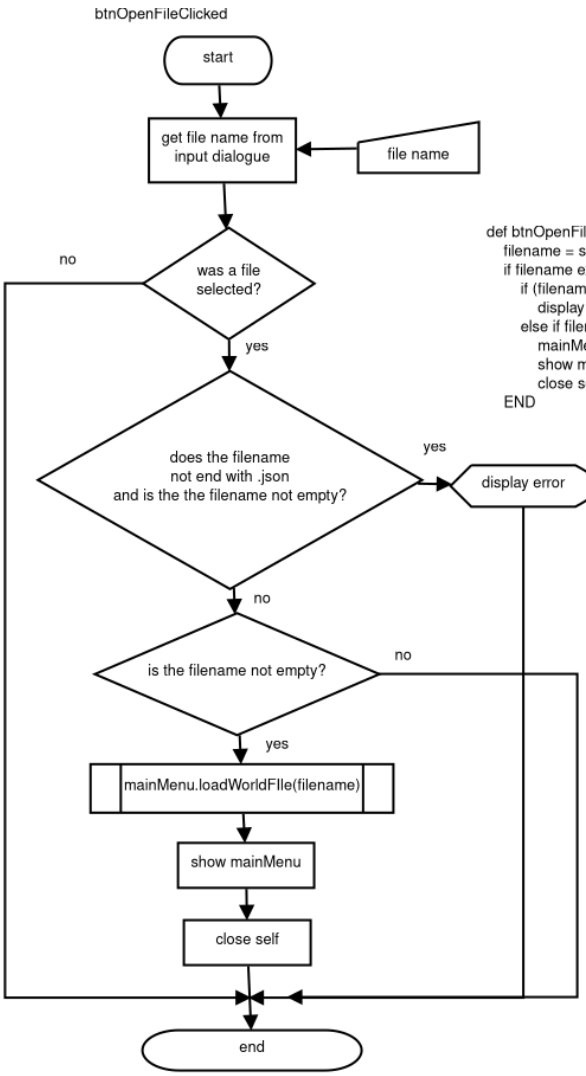
1 Implementing the Solution

1.1 Algorithms and Psuedocode

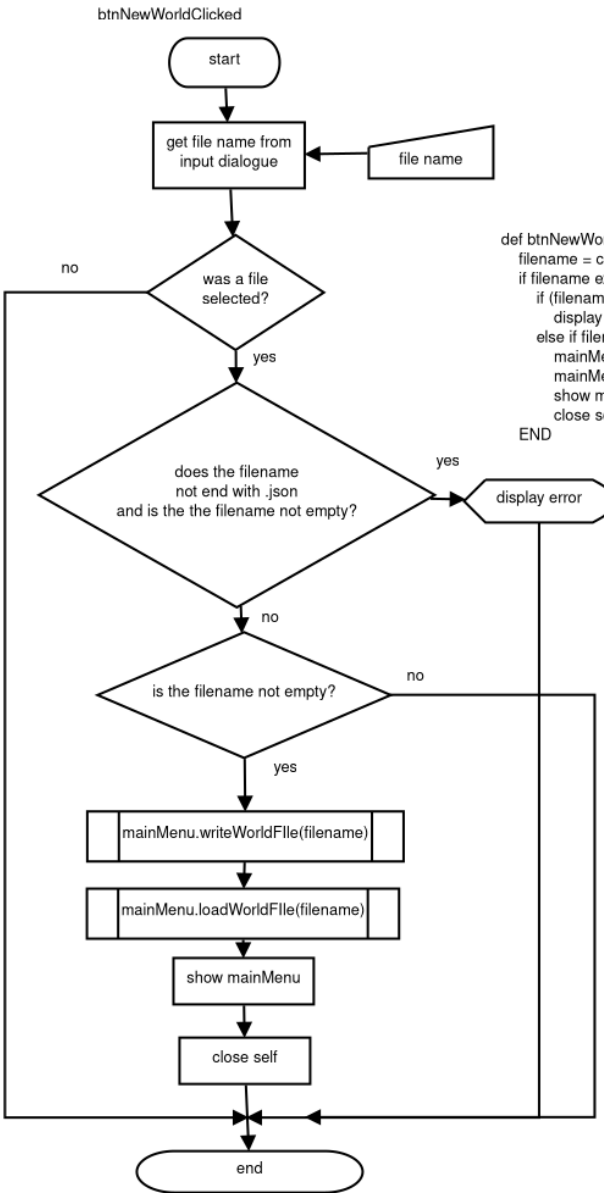
1.1.1 introForm

```
def init():
    FILE_PATH = python file directory path
    load UI from UI File
    variables = default value
    connect buttons and actions to subroutines
    END
```



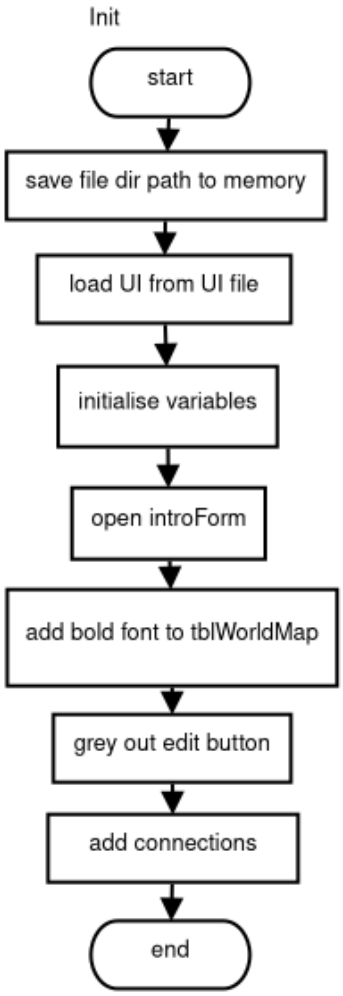



```
def btnOpenFileClicked():
    filename = select file dialogue()
    if filename exists:
        if (filename doesn't end with ".json") and (filename isn't empty):
            display error
        else if filename isn't empty:
            mainMenu.loadWorldFile(filename)
            show mainMenu
            close self
    END
```

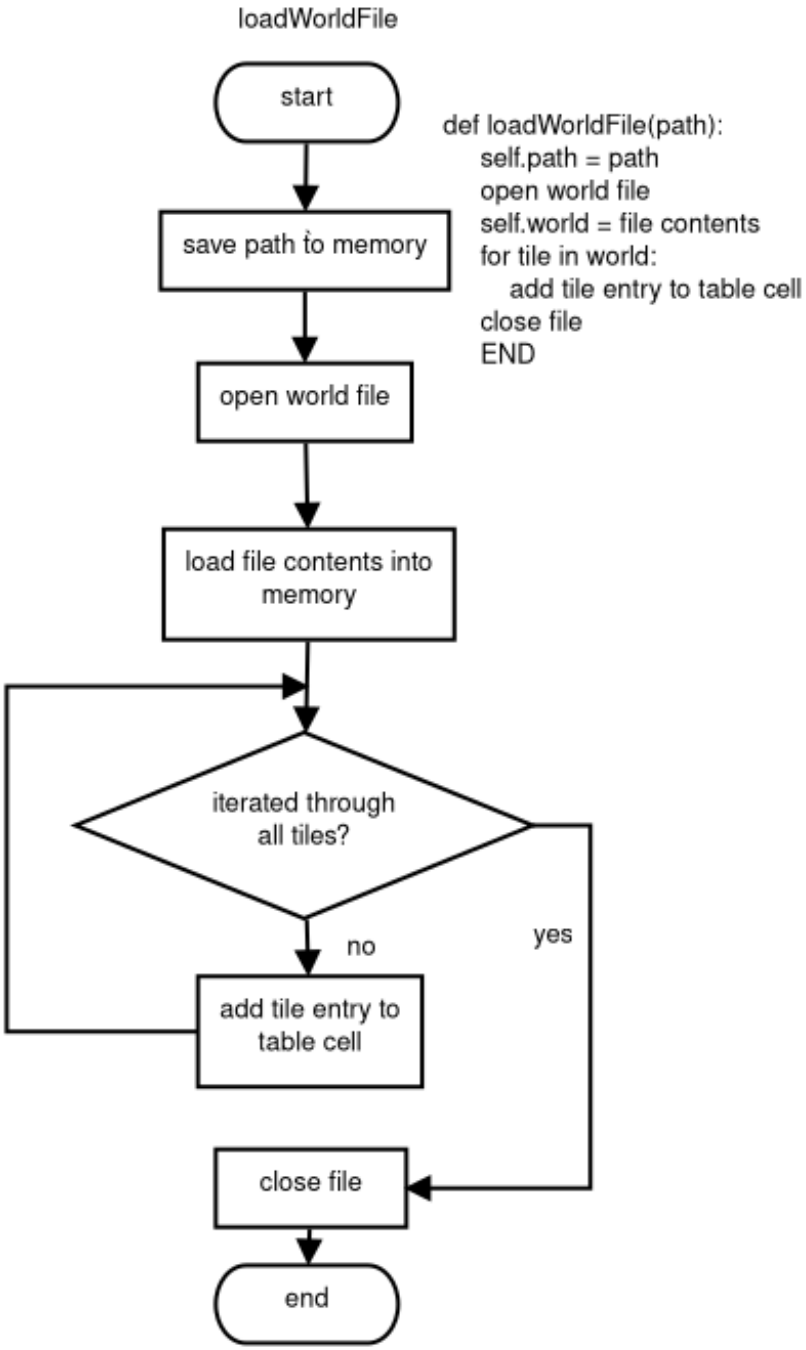



```
def btnNewWorldClicked():
    filename = create file dialogue()
    if filename exists:
        if (filename doesn't end with ".json") and (filename isn't empty)
            display error
        else if filename isn't empty:
            mainMenu.writeWorldFile(filename)
            mainMenu.loadWorldFile(filename)
            show mainMenu
            close self
    END
```

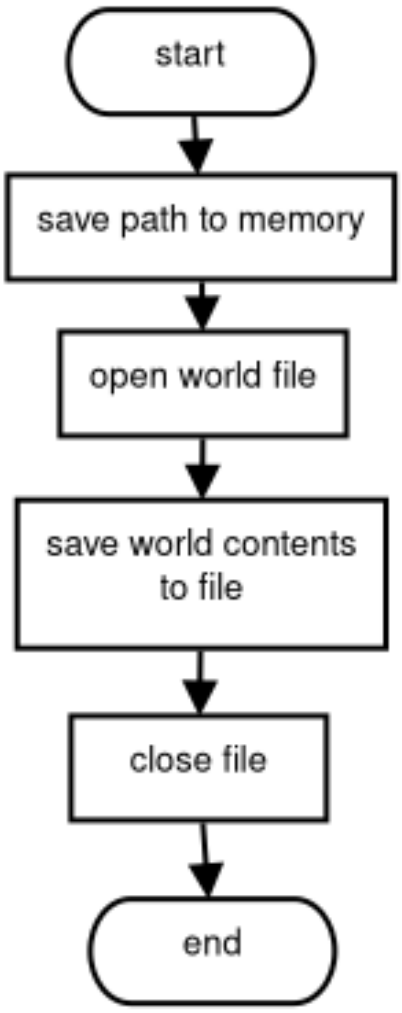
1.1.2 mainMenuForm



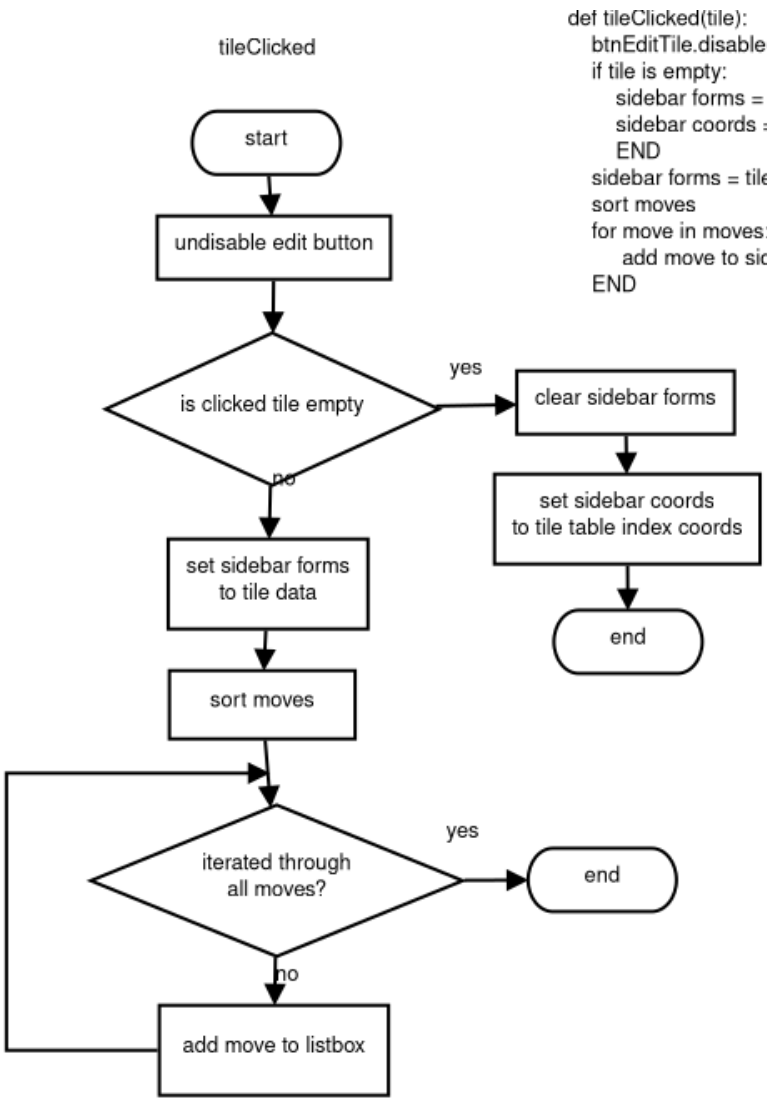

```
def init():
    FILE_PATH = python file directory path
    load UI from UI File
    variables = default value
    introForm.open
    tblWorldMap.setBold(true)
    btnEditTile.disable()
    connect buttons to subroutines
    END
```


loadWorldFile
start
save path to memory
open world file
load file contents into memory
iterated through all tiles?
no
yes
add tile entry to table cell
close file
end
def loadWorldFile(path):
self.path = path
open world file
self.world = file contents
for tile in world:
add tile entry to table cell
close file
END

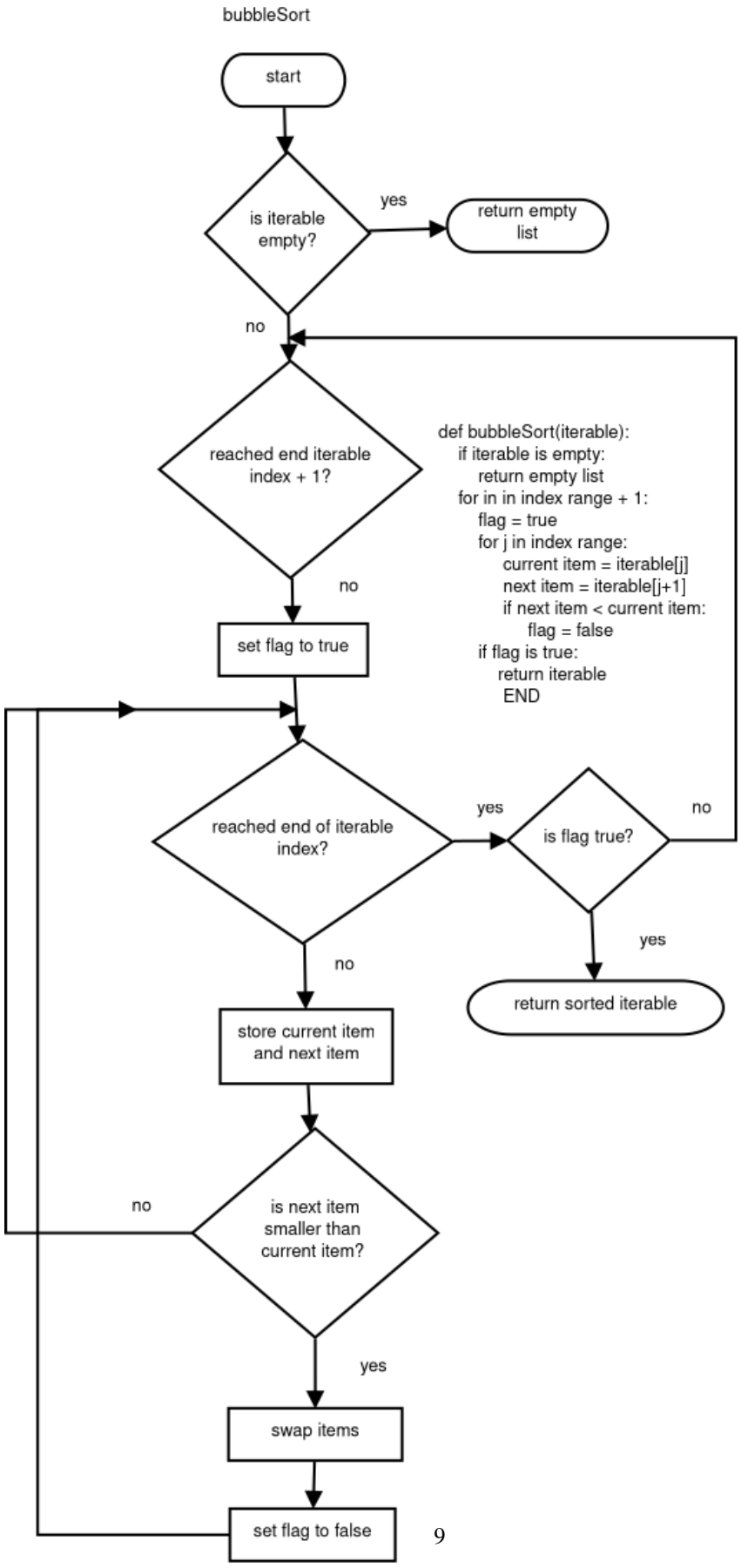
writeWorldFile



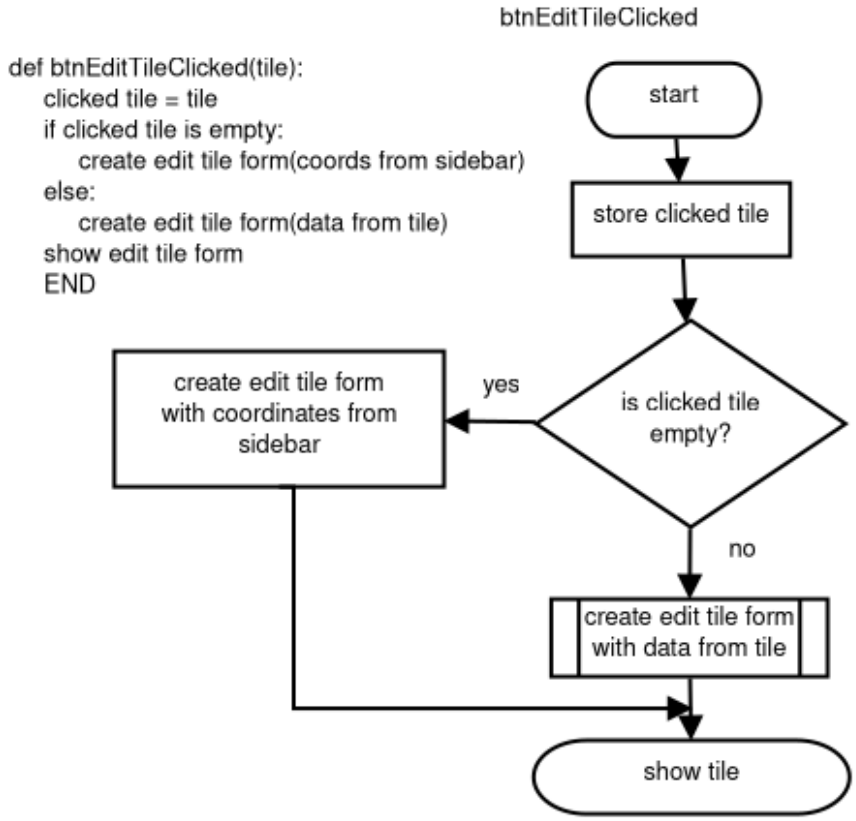

```
def writeWordFile(path):
    self.path = path
    open world file
    save self.world to file
    close file
    END
```

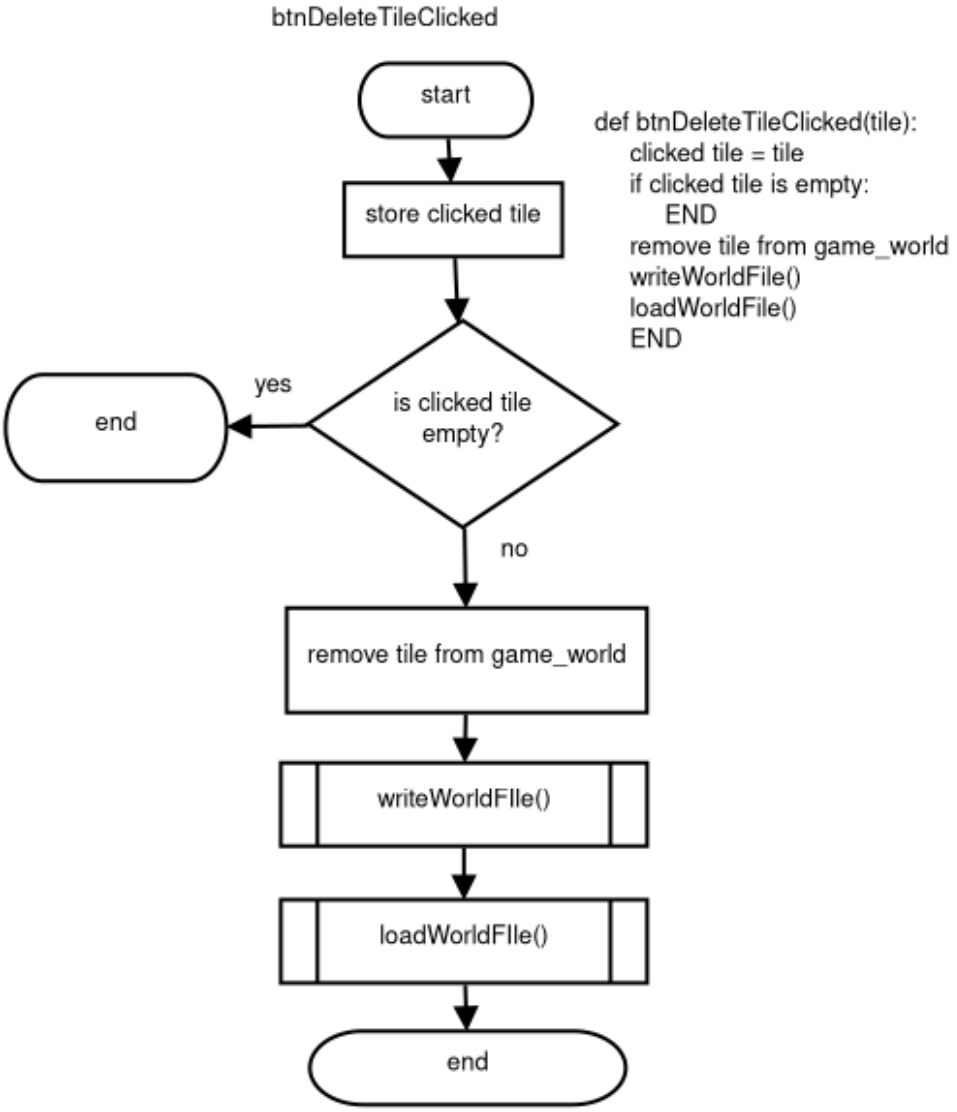


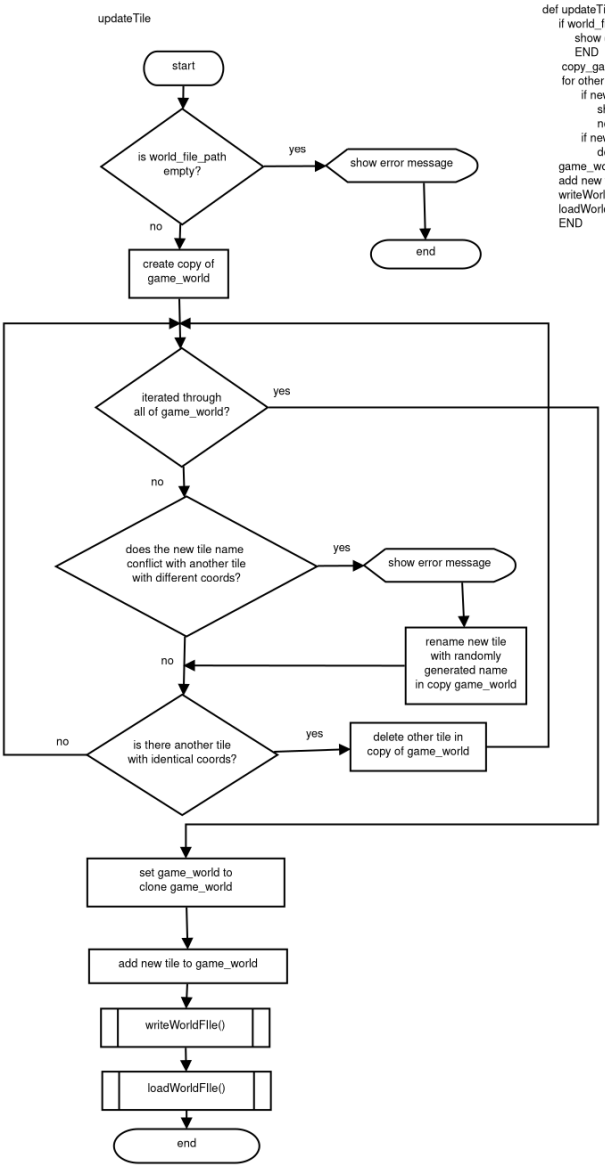

```
def tileClicked(tile):
    btnEditTile.disable(false)
    if tile is empty:
        sidebar forms = nothing
        sidebar coords = tile coords
        END
    sidebar forms = tile data
    sort moves
    for move in moves:
        add move to sidebar listbox
    END
```


bubbleSort
start
is iterable empty?
yes
return empty list
no
reached end iterable index + 1?
no
set flag to true
def bubbleSort(iterable):
if iterable is empty:
return empty list
for in in index range + 1:
flag = true
for j in index range:
current item = iterable[j]
next item = iterable[j+1]
if next item < current item:
flag = false
if flag is true:
return iterable
END
reached end of iterable index?
yes
is flag true?
no
yes
return sorted iterable
no
store current item and next item
is next item smaller than current item?
no
yes
swap items
set flag to false

```
def btnEditTileClicked(tile):
    clicked tile = tile
    if clicked tile is empty:
        create edit tile form(coords from sidebar)
    else:
        create edit tile form(data from tile)
    show edit tile form
    END
```



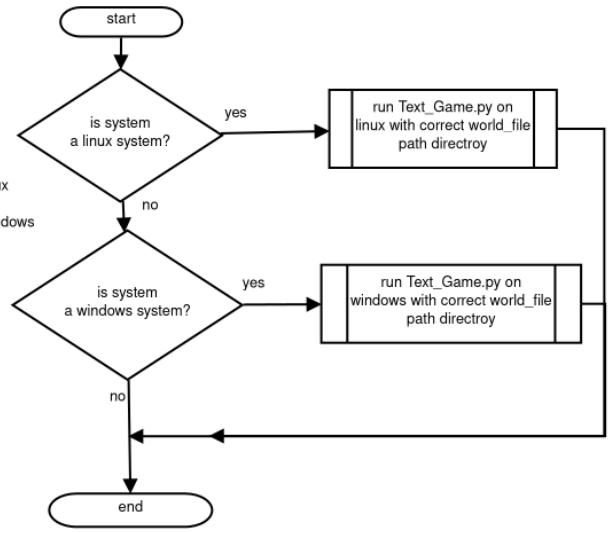

btnDeleteTileClicked
start
store clicked tile
is clicked tile empty?
yes
end
no
remove tile from game_world
writeWorldFIle()
loadWorldFIle()
end
def btnDeleteTileClicked(tile):
clicked tile = tile
if clicked tile is empty:
END
remove tile from game_world
writeWorldFile()
loadWorldFile()
END




```
def updateTile(new tile):
    if world_file_path is empty:
        show error message
        END
    copy_game_world = copy(game_world)
    for other tile in game_world:
        if new_tile name == other tile name AND new_tile coords != other tile coords
            show error message
            new tile name = randomly generated name
        if new_tile coords == other tile coords:
            delete other  tile in copy_game_world
    game_world = copy_game_world
    add new tile to game_world
    writeWorldFile()
    loadWorldFile()
    END
```

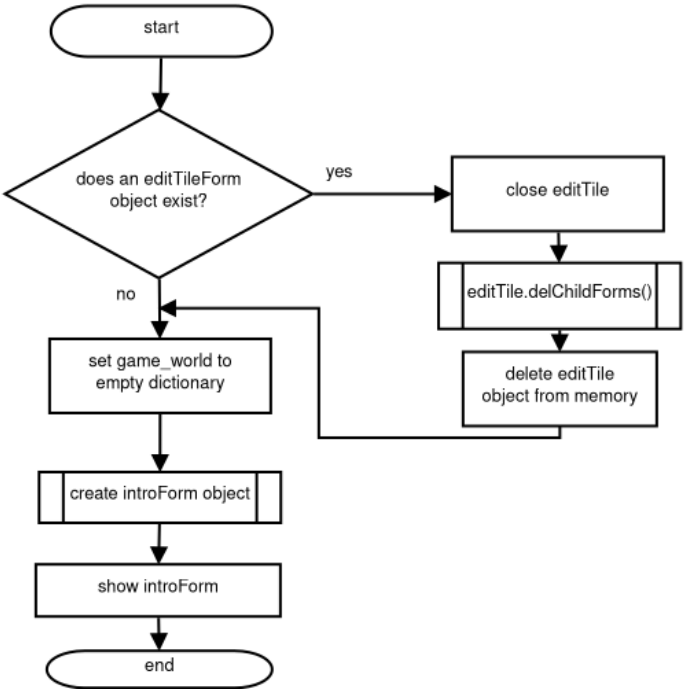
btnPlayGameClicked

```
def btnPlayGameClicked():
    if system is a linux system:
        run Text_Game.py + correct dir path on linux
    if system is a windows system:
        run Text_Game.py + correct dir path on windows
    END
```



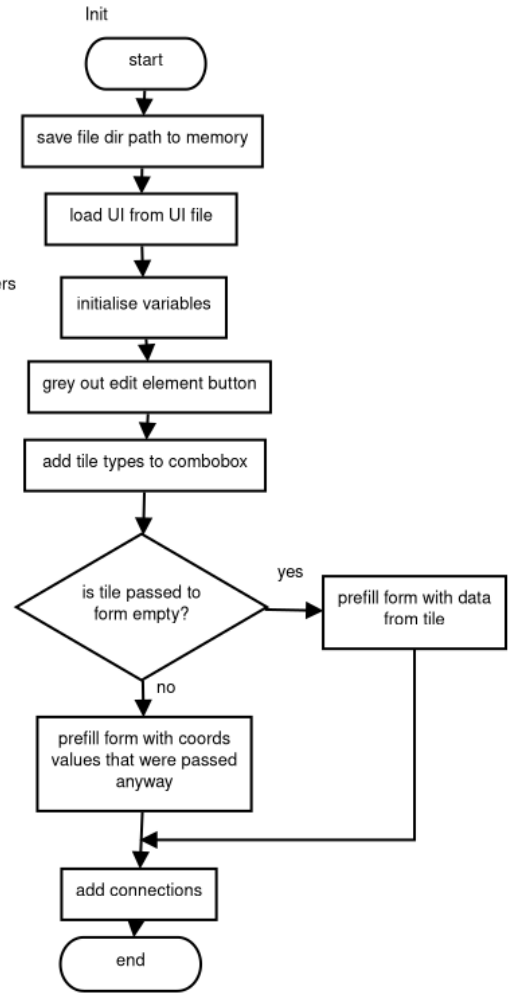
openNewFile

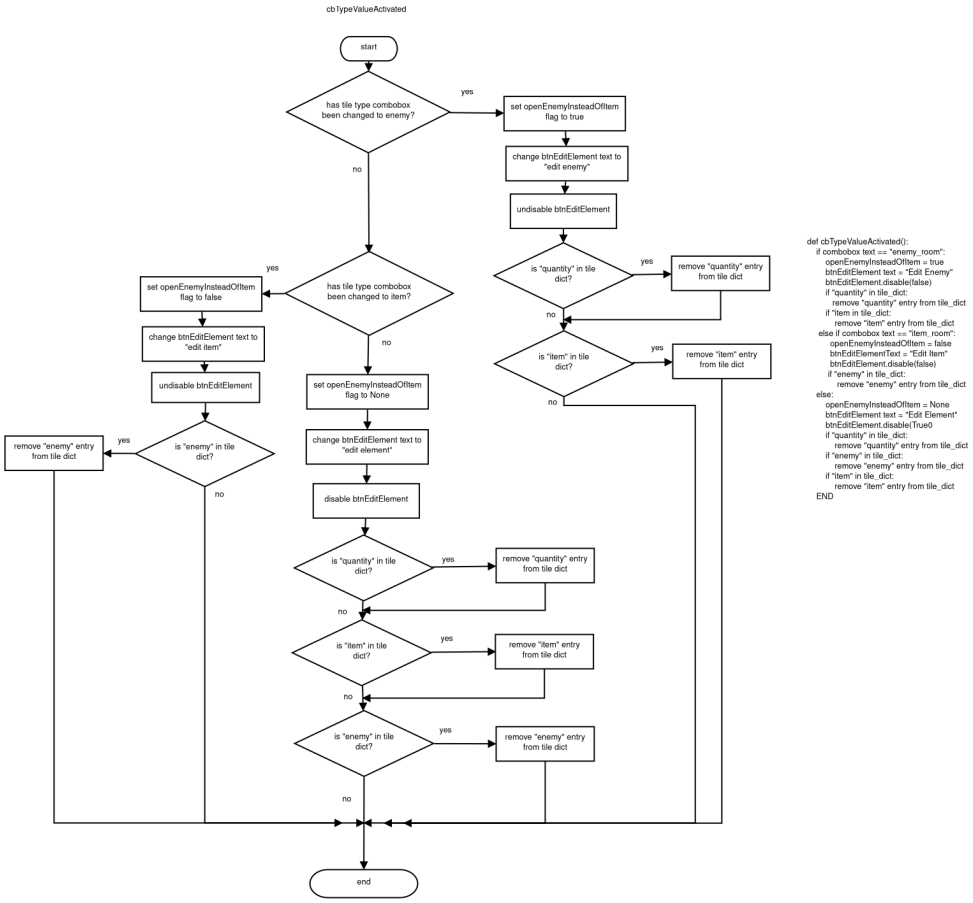
```
def openNewFile():
    if editTileForm object exists:
        close editTile
        editTile.delChildForms()
        delete editTile object
    game_world = {}
    introForm = introForm()
    show introForm
    END
```

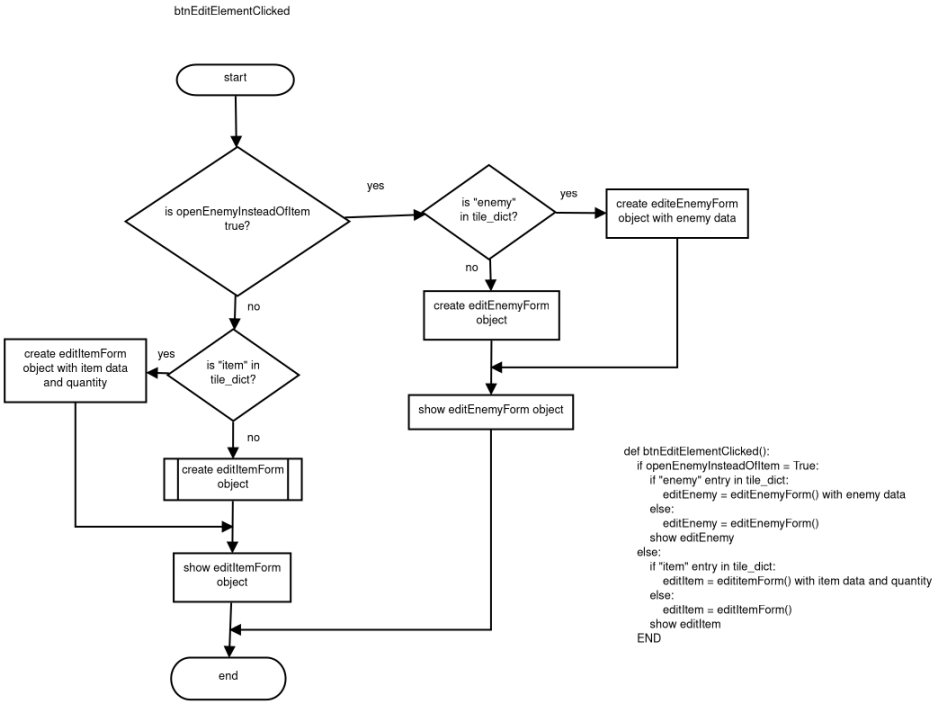


1.1.3 editTileForm

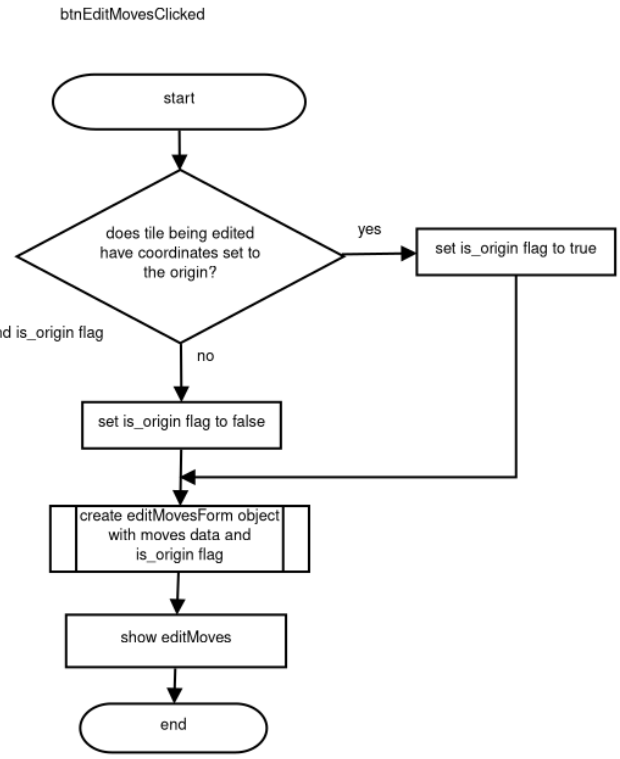
```
def init(tile data, coords):
    FILE_PATH = python file directory path
    load UI from UI File
    variables = default value
    btnEditElement.disable()
    add tile types to comb
    if tile passed to form is empty:
        prefill form with data from tile structure
    else:
        prefill form with coords values passed as parameters
    connect buttons and actions to subroutines
    END
```




cbTypeValueActivated
start
has tile type combobox been changed to enemy?
yes
set openEnemyInsteadOfItem flag to true
change btnEditElement text to "edit enemy"
undisable btnEditElement
no
has tile type combobox been changed to item?
yes
set openEnemyInsteadOfItem flag to false
change btnEditElement text to "edit item"
undisable btnEditElement
no
set openEnemyInsteadOfItem flag to None
change btnEditElement text to "edit element"
disable btnEditElement
is "enemy" in tile dict?
yes
remove "enemy" entry from tile dict
no
is "quantity" in tile dict?
yes
remove "quantity" entry from tile dict
no
is "item" in tile dict?
yes
remove "item" entry from tile dict
no
is "enemy" in tile dict?
yes
remove "enemy" entry from tile dict
no
is "quantity" in tile dict?
yes
remove "quantity" entry from tile dict
no
is "item" in tile dict?
yes
remove "item" entry from tile dict
no
end
def cbTypeValueActivated():
if combobox text == "enemy_room":
openEnemyInsteadOfItem = true
btnEditElement text = "Edit Enemy"
btnEditElement.disable(false)
if "quantity" in tile_dict:
remove "quantity" entry from tile_dict
if "item in tile_dict:
remove "item" entry from tile_dict
else if combobox text == "item_room":
openEnemyInsteadOfItem = false
btnEditElementText = "Edit Item"
btnEditElement.disable(false)
if "enemy" in tile_dict:
remove "enemy" entry from tile_dict
else:
openEnemyInsteadOfItem = None
btnEditElement text = "Edit Element"
btnEditElement.disable(True0
if "quantity" in tile_dict:
remove "quantity" entry from tile_dict
if "enemy" in tile_dict:
remove "enemy" entry from tile_dict
if "item" in tile_dict:
remove "item" entry from tile_dict
END

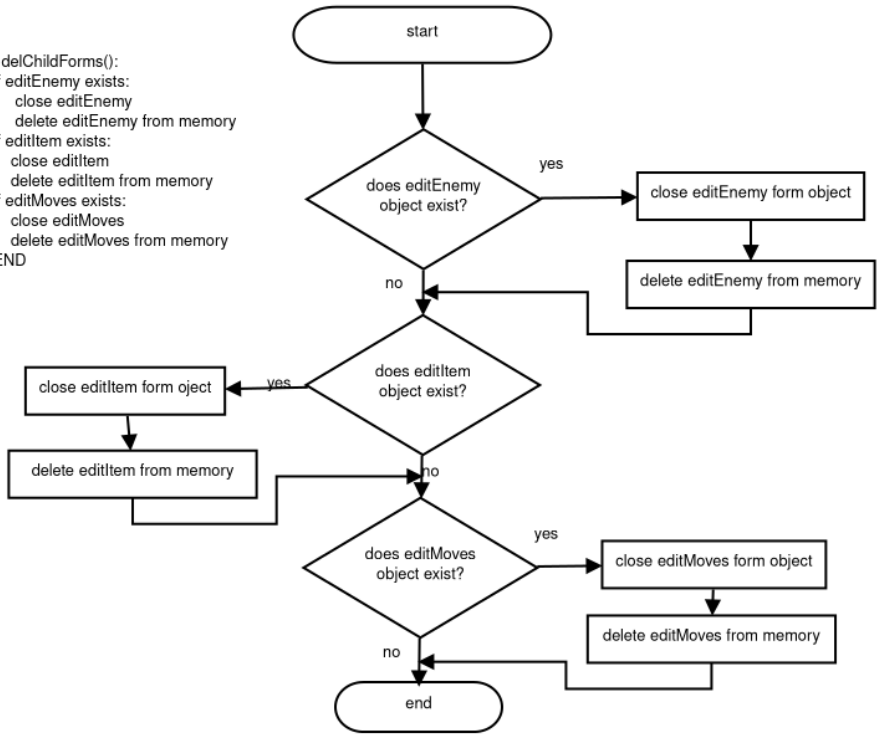

btnEditElementClicked
start
is openEnemyInsteadOfItem true?
yes
is "enemy" in tile_dict?
yes
create editeEnemyForm object with enemy data
no
create editEnemyForm object
no
is "item" in tile_dict?
yes
create editItemForm object with item data and quantity
no
create editItemForm object
show editEnemyForm object
show editItemForm object
end
def btnEditElementClicked():
if openEnemyInsteadOfItem = True:
if "enemy" entry in tile_dict:
editEnemy = editEnemyForm() with enemy data
else:
editEnemy = editEnemyForm()
show editEnemy
else:
if "item" entry in tile_dict:
editItem = editItemForm() with item data and quantity
else:
editItem = editItemForm()
show editItem
END

```
def btnEditMovesClicked():
    if coords input boxes are at 0,0:
        is_origin = true
    else:
        is_origin = false
    editMoves = editMovesForm() with moves data and is_origin flag
    show editMoves
    END
```



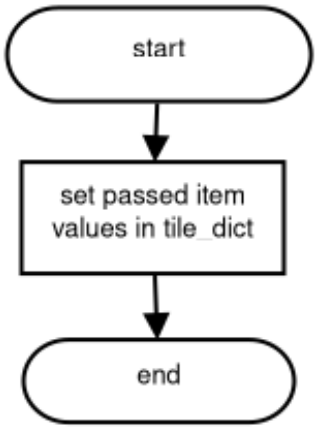
delChildForms

```
def delChildForms():
   if editEnemy exists:
      close editEnemy
      delete editEnemy from memory
   if editItem exists:
      close editItem
      delete editItem from memory
   if editMoves exists:
      close editMoves
      delete editMoves from memory
   END
```



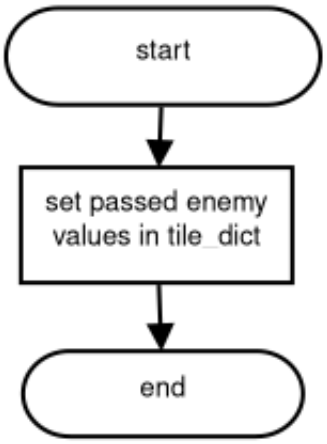
updateItem

```
def updateItem(values):
   tile dict item entry = passed values
   END
```



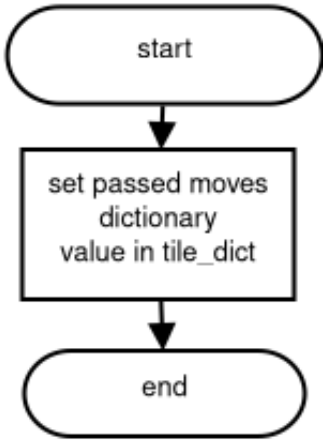
updateEnemy

```
def updateEnemy(values):
    tile dict enemy entry = passed values
    END
```

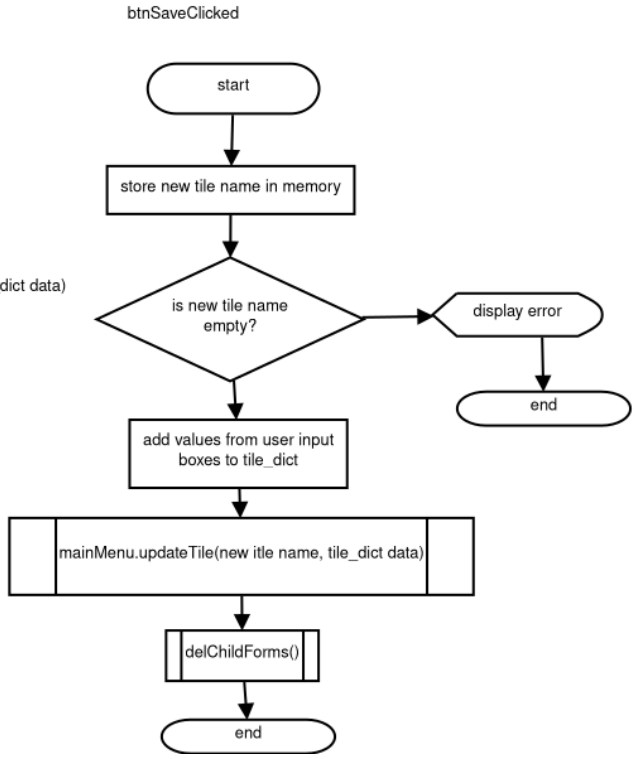


updateMoves

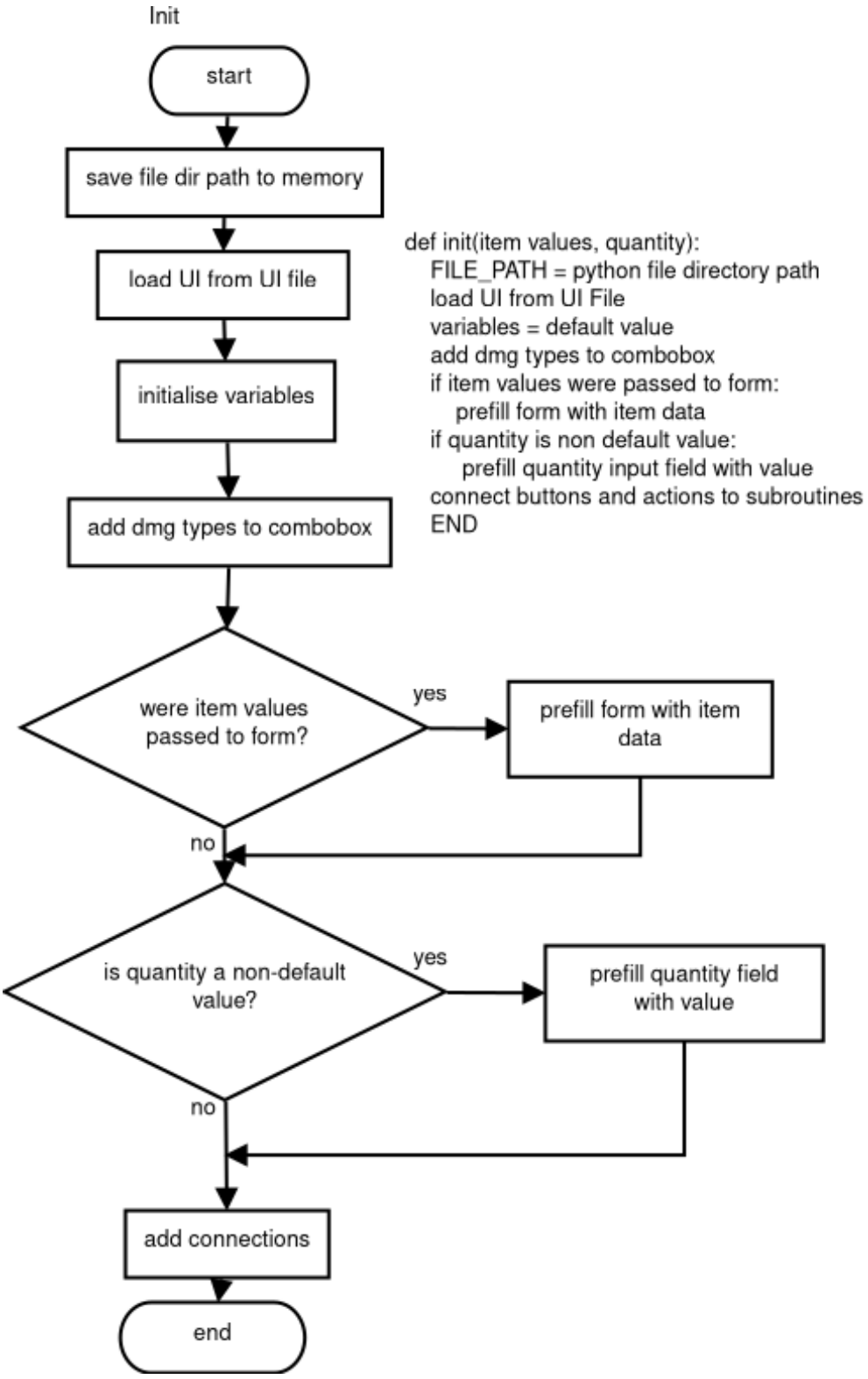
```
def updateMoves(moves_dict):
    tile dict moves_dict entry = passed moves_dict
    END
```

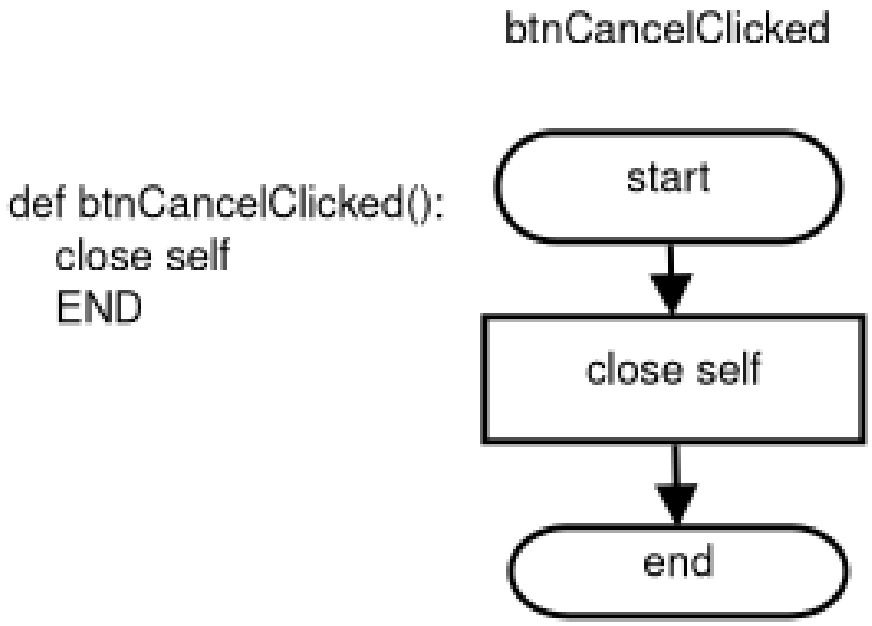


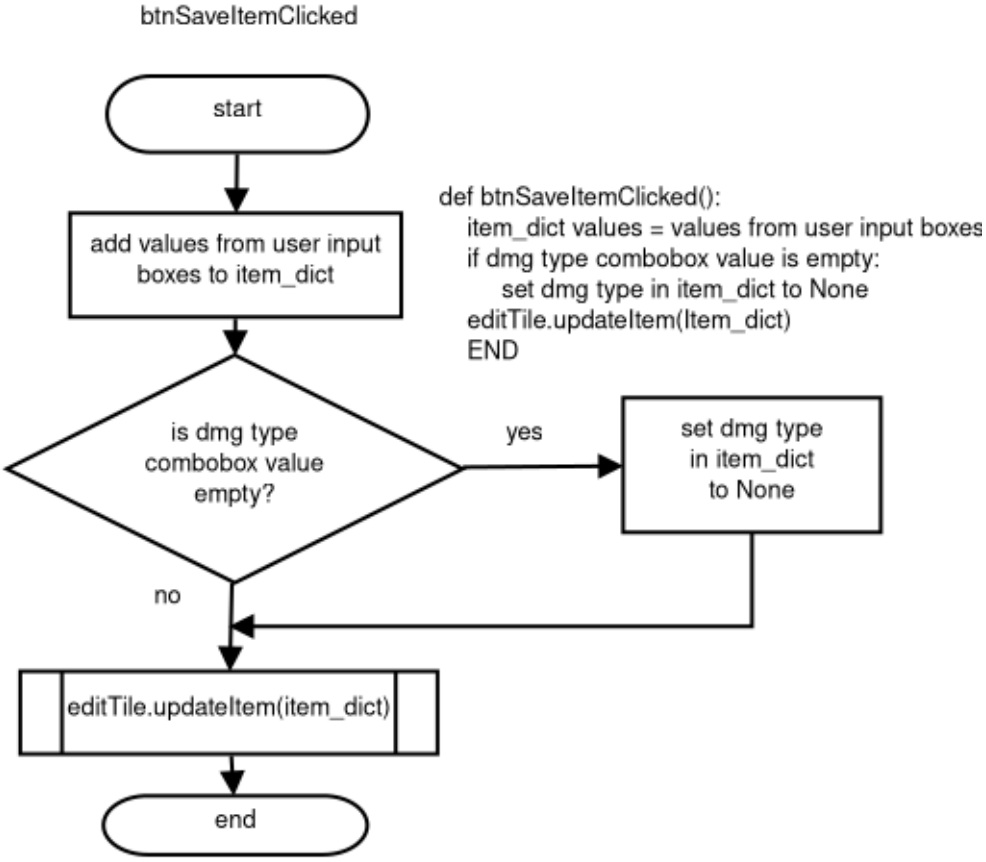
```
def btnSaveClicked():
    new tile name = value from user input
    if new tile name is empty:
        display error message
        END
    tile_dict values = values from user input
    mainMenu.updateTile(new tile name, tile_dict data)
    delChildForms()
    END
```



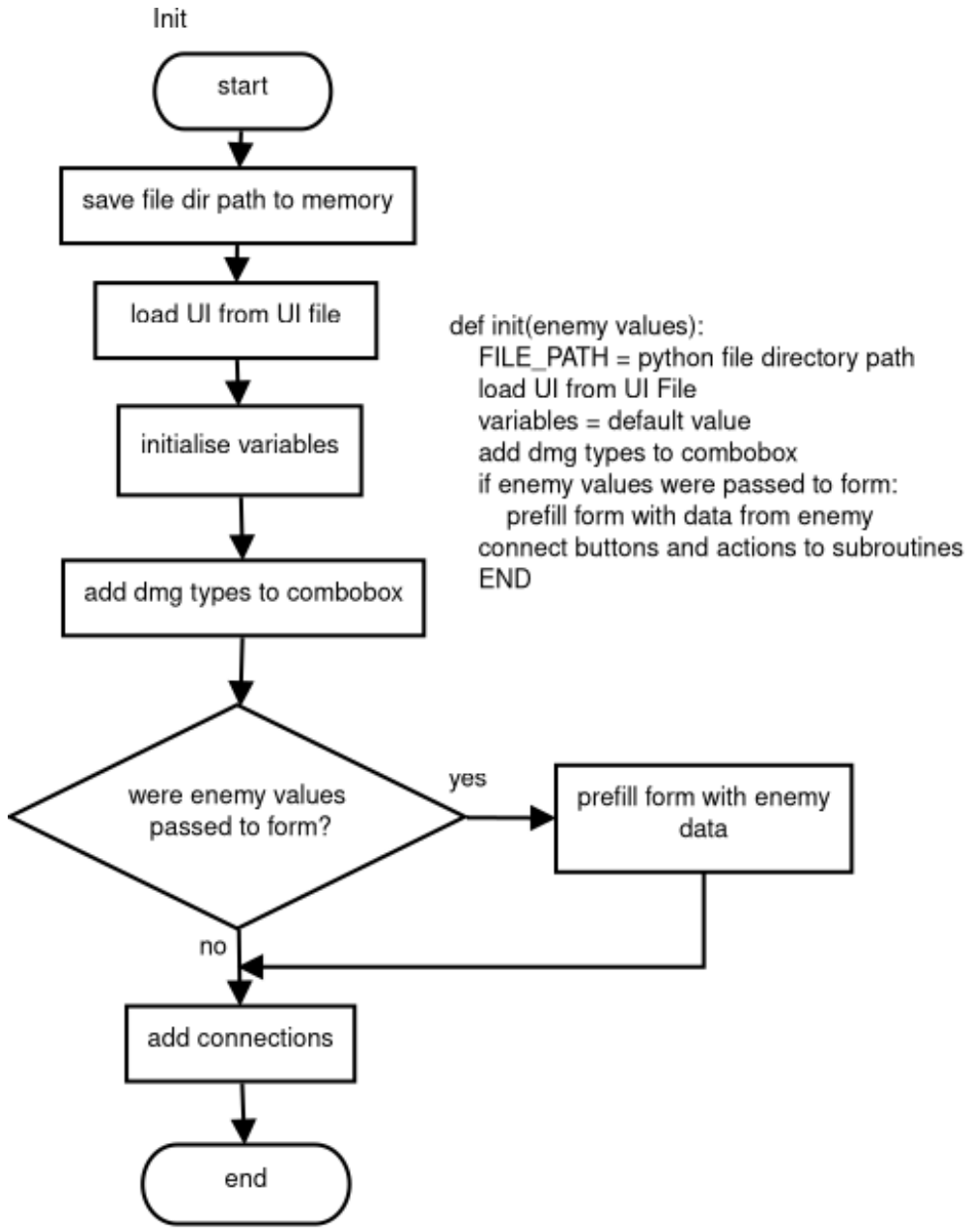
1.1.4 editItemForm




btnCancelClicked
def btnCancelClicked():
close self
END
start
close self
end

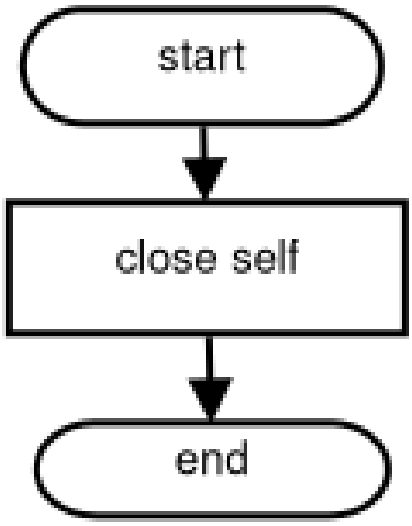

btnSaveItemClicked
start
add values from user input boxes to item_dict
is dmg type combobox value empty?
yes
set dmg type in item_dict to None
no
editTile.updateItem(item_dict)
end
def btnSaveItemClicked():
item_dict values = values from user input boxes
if dmg type combobox value is empty:
set dmg type in item_dict to None
editTile.updateItem(Item_dict)
END

1.1.5 editEnemyForm



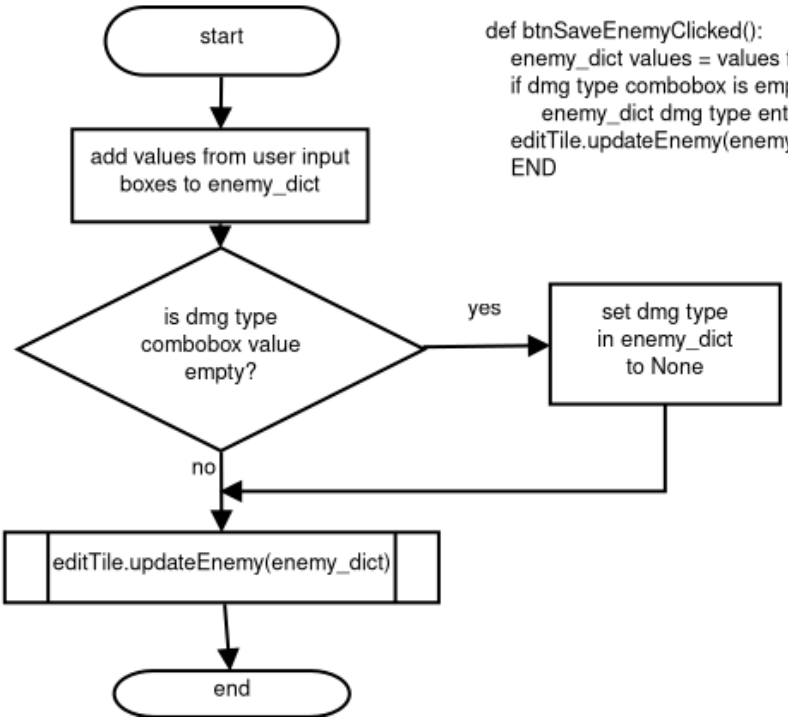
btnCancelClicked

```
def btnCancelClicked():
    close self
    END
```



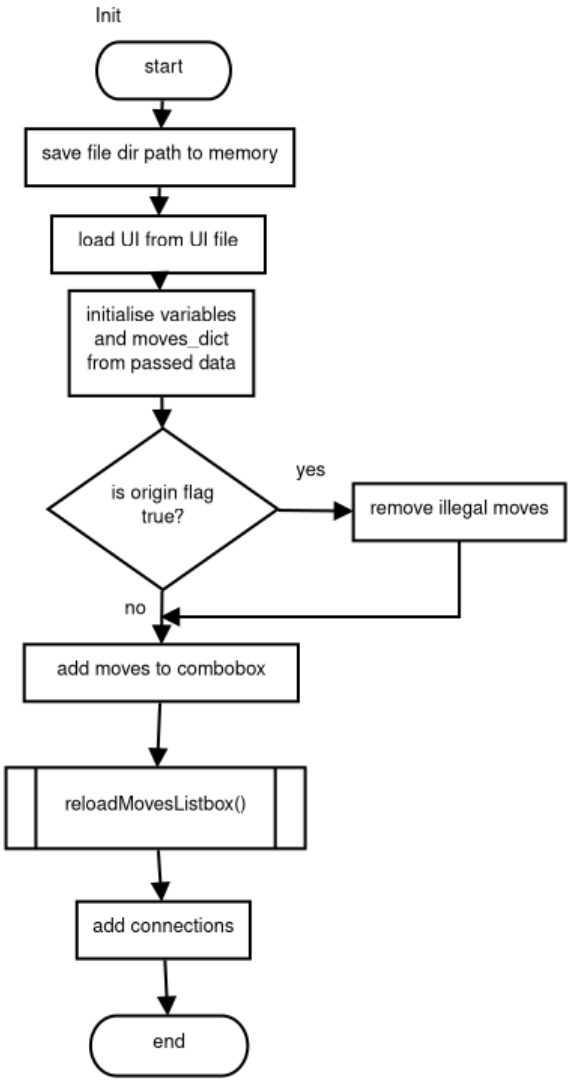
btnSaveEnemyClicked

```
def btnSaveEnemyClicked():
    enemy_dict values = values from user input boxes
    if dmg type combobox is empty:
        enemy_dict dmg type entry = None
    editTile.updateEnemy(enemy_dict)
    END
```

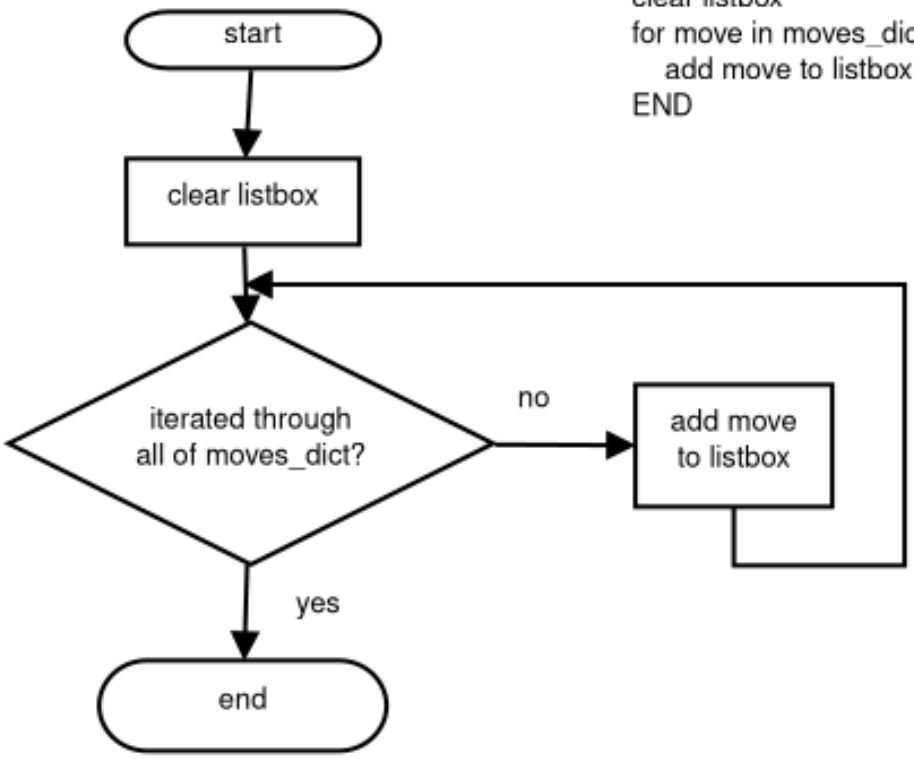


1.1.6 editMovesForm

```
def init(old moves_dict, origin flag):
    FILE_PATH = python file directory path
    load UI from UI File
    variables = default values
    moves_dict = old moves_dict
    if origin_flag is true:
        remove illegal moves
    add moves to combobox
    connect buttons and actions to subroutines
    END
```

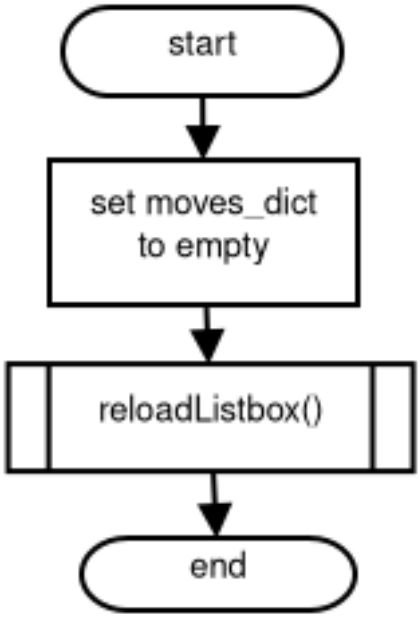


reloadListbox



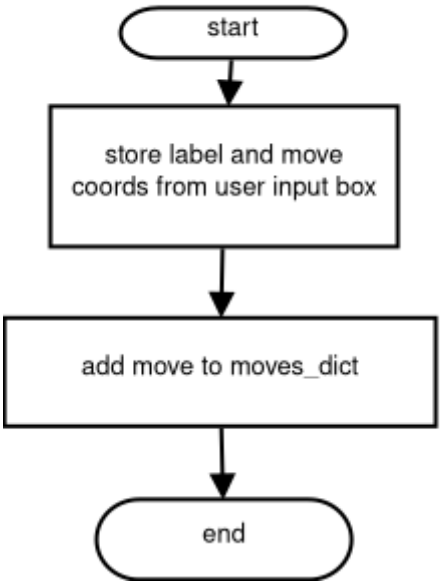

```
def reloadListbox():
    clear listbox
    for move in moves_dict:
        add move to listbox
    END
```

btnClearClicked




```
def btnClearlClicked()
    moves_dict = {}
    reloadListbox()
    END
```

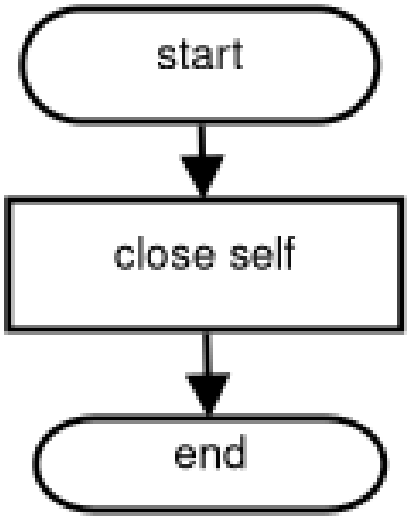
btnAddMoveClicked

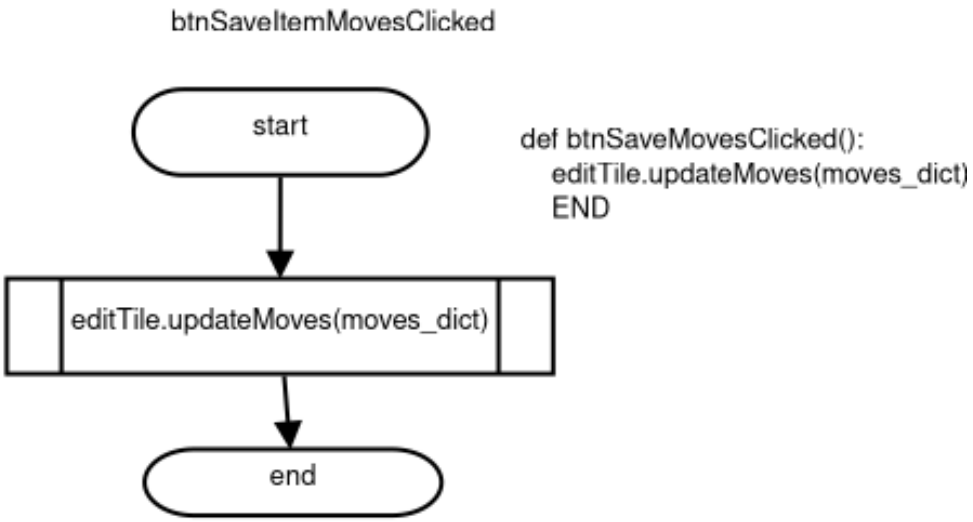



```
def reloadListbox():
    label = value from user input box
    move coords = value from user input box
    add complete move to moves_dict
    END
```

btnCancelClicked

```
def btnCancelClicked()
    close self
    END
```




btnSaveItemMovesClicked
start
editTile.updateMoves(moves_dict)
end
def btnSaveMovesClicked():
editTile.updateMoves(moves_dict)
END

1.2 Source Code and User Interface

Please see: `https://github.com/20dj-kws-sdd/tbrpggepp`

1.3 User Manual

Please see: `https://github.com/20dj-kws-sdd/tbrpggepp/raw/master/manual/manual.pdf`

2 Testing and Maintaining the Solution

2.1 Test Plan and Report

The testing conducted during the development of the program consisted of methodically checking the function of each routine (module-level testing), the methods in which the routines interacted with each other and the program as a whole (program-level testing), and running the program on multiple different operating system environments and checking correct functioning (system-level testing). This was done by enumerating through the permutations of possible input into specific modules and checking if they functioned correctly, and then checking if the modules integrated with each other successfully throughout usage of the program. Near the end of the development phase the program's functionality was checked on a Windows machine as well as the Linux machine the program was developed on, ensuring proper system-level testing. Throughout the development cycle, individual alpha testing was conducted most frequently. At periods late in the development cycle small-scale beta testing was conducted as well, in order to gauge UI effectiveness and the intuitiveness of the program.

The range of test data used consisted of game-world json files that had been created prior to undergoing this project. The test data ranged from quite small (0 to 6 tiles) to reasonably large (18 full tiles, sufficient for load testing) and was manipulated via operation of the program to enumerate through possible methods of data transformation and ensure continual correct functioning despite the range of possible changes made via program usage. The cross-platform methods for data input, output and transformation were found to be sufficient via system-level testing.

The table below contains details of the testing that took place for all modules that involved the input or transformation of data.

Item being tested	Test Data being used	Reason for inclusion	Expected Result	Pass/Fail
btnOpenFileClicked in introForm	non-json file pathname	Test case where user selects invalid file	Error message and no other change	Pass
btnOpenFileClicked in introForm	empty file pathname	Test case where user cancels file dialog, appearing programmatically as an empty file	No effect	Pass
btnOpenFile Clicked in introForm	valid json file pathname	Ensure ordinary operation of module	mainMenu form opens and loadWorldFile is ran	Pass
btnNewWorldFile Clicked in introForm	non-json file pathname	Test case where user creates invalid file	Error message and no other change	Pass
btnNewWorldFile Clicked in introForm	empty file pathname	Test case where user cancels file dialog, appearing programmatically as an empty file	No effect	Pass
btnNewWorldFile Clicked in introForm	valid json file pathname	Ensure ordinary operation of module	mainMenu form opens and writeWorldFile, then loadWorldFile is ran	Pass
loadWorldFile in mainMenu	Empty world file (0 tiles)	Test lower bound for data input	Grid appears empty but remains functional	Pass
loadWorldFile in mainMenu	Small world file (6 tiles)	Test reasonable amount of data input	Tiles load successfully into mainForm	Pass

loadWorldFile in mainMenu	Large world file (18 tiles)	Load testing for data input	Tiles load successfully into mainForm	Pass
writeWorldFile in mainMenu	World file paths of varying locations	Test proper functionality of module	Creates file in specified path and dumps game-world contents into it	Pass
tileClicked in mainMenu	Populated game-world tile indexes	Test proper functionality of module when clicking populated tiles	Relevant sidebar information appears to the right, Edit Tile button ungreys if previously grey	Pass
tileClicked in mainMenu	Empty game-world tile indexes	Test proper functionality of module when clicking empty tiles	Only coord values are updated in sidebar information, other fields appear blank. Edit Tile button ungreys if previously grey.	Pass
bubbleSort in mainMenu	arbitrary moves_dict labels, pulled from populated tile data	Test proper functionality of module	Returns sorted labels, displayed in mainMenu sidebar	Pass
btnEditTileClicked in mainMenu	Empty tile selected	Test proper functionality of module when attempting to edit empty tiles	Edit Tile form opens with only coordinate values prefilled	Pass
btnEditTileClicked in mainMenu	Populated tile selected	Test proper functionality of module when attempting to edit populated tiles	Edit Tile form opens with all values prefilled	Pass
btnDeleteTileClicked in mainMenu	Populated tile selected	Test proper functionality of module when attempting to delete populated tiles	Populated tile in game-world grid is replaced with an empty tile	Pass
btnDeleteTileClicked in mainMenu	Empty tile selected	Test proper functionality of module when attempting to delete empty tiles	No change	Pass
updateTile in mainMenu	world file path is empty	Check that an error is caught if the user tries to modify a game-world when in fact no game-world is being edited	Error message displayed and no change applied	Pass
updateTile in mainMenu	tile data where the new tile name overlaps with a preexisting tile in a different location	Check whether the program overwrites the older conflicting tile against the user's will or not.	Error message displayed, new tile renamed to a randomly generated name, no tiles are overwritten	Pass

updateTile in mainMenu	tile data where the new tile's coordinates overlap with a preexisting tile in the same location	Check whether the program follows the user's intention of updating a preexisting tile with new data	No error message displayed, older tile is overwritten with new tile data	Pass
updateTile in mainMenu	ordinary tile data without any conflicting overlap in name or coordinates	Ensure proper functioning of module in ordinary situation	New populated tile created at specified coordinates	Pass
editTileForm module	tile data from a populated tile	Ensure ordinary operation of module when passed a populated tile	Input fields prefilled with tile data	Pass
editTileForm module	tile data from an unpopulated tile	Ensure ordinary operation of module when passed an empty tile	Only coorindate Input field prefilled, other fields left blank	Pass
editTileForm module	item/enemy/moves data	Ensure ordinary operation of module when given data from sub-form	Item/Element/Moves data is saved in local memory	Pass
editItemForm module	item data from a populated tile	Ensure ordinary operation of module when passed populated item data	Input fields prefilled with item data	Pass
editItemForm module	empty item data	Ensure ordinary operation of module when not passed any item data	Input fields left blank	Pass
editEnemyForm module	enemy data from a populated tile	Ensure ordinary operation of module when passed populated enemy data	Enemy data prefilled in input fields	Pass
editEnemyForm module	empty enemy data	Ensure ordinary operation of module when not passed any enemy data	Input fields left blank	Pass
editMovesForm module	moves_dict data from a populated tile	Ensure ordinary operation of module when passed populated moves_dict data	Moves data prefilled in input fields and listbox displays moves	Pass
editMovesForm module	empty moves_dict data	Ensure ordinary operation of module when not passed any moves_dict data	Input fields and listbox left blank	Pass

2.2 Maintenance Overview

Three possible updates or improvements to the project could be as follows:

1. Integrating version control for world-files into program
   Additional forms for committing, reverting or branching changes made to the game-world file into a local git repo would greatly improve the user's experience in managing changes made to the game-world throughout the operation of the program. This feature was initially within the program's intended scope, but was left out after realizing it didn't affect the core functionality of the program and the actions performed by the additional forms could still be performed by the user by operating the git program manually.

2. Adding functionality for deleting specific moves in the editMoves form
   In the case of the user making an error in the creation of moves for their tile, a "clear all" button is provided - which inevitably would create annoyance for the user since they may temporarily lose their work on the creation of additional valid moves in the tile. Therefore the feature of removing individual moves from the tile's moves_dict structure would aid them greatly from a quality-of-life perspective in the usage of the program. This feature was left out of the original scope of the project due to a) apparent difficulty to implement and b) the user may, with some care, take steps to mitagate this data loss such as saving the editMoves form every time a valid move is added and cancelling the form instead of clearing it when a mistake is made.

3. A search bar for tiles in the mainMenu form
   A search bar for tile names in the mainMenu form would assist the user in locating elements of the game-world grid in the case they add to it to the point of vast size. This too would be a useful quality-of-life feature but was left out during implementation (despite being initially within the scope of the project) due to difficulty of implementing and non-essential nature to the functioning of the program.

3 Documentation and Project Work

3.1 Learning Journal

Please see: `https://20dj-kws-sdd.github.io/`

3.2 Project Work