Text Adventure Editor Pseudocode

* **main()** -
  + Runs the main loop
  + Calls a menu
  + Sends control to other parts of the program
  + Handles invalid input from menu

Start a while loop, give keepGoing “True”

Call getMenuChoice(), with the parameter {choice}

If choice == “0”

Give keepGoing “False”

Print “See you later!”

If choice == “1”

gameFile gets getDefaultGame()

If choice == “2”

Call loadGame()

If choice == “3”

Call saveGame()

If choice == “4”

Call editNode()

If choice == “5”

Call playGame

Else:

Print “That’s not a valid option! Please try again.”

* **getMenuChoice()**
  + prints a menu of user options
  + returns a menu choice

Print the menu

“Welcome to the game editor! What would you like to do?

1. Exit
2. Load the default game
3. Load a game file
4. Save the current game
5. Edit or add a node
6. Play the current game”

The variable {choice} will get the response to “Enter here: “

Return {choice}

* **playGame()**
  + plays the game
  + Keeps going until next node is "quit"

Create a variable keepGoing, which gets “True”

Make a while loop with keepGoing:

Call playNode(), give it the game data and the current node (parameter {currentNode})

If currentNode is equal to “quit”, keepGoing gets “False”

If the player’s answer is “1”:

currentNode gets the third item from the current node

If the player’s answer is “2”:

currentNode gets the fifth item from the current node

* **playNode()**
  + given the game data and a node,
  + plays out the node
  + returns the next node

give keepGoing True

while keepGoing == true:

Print the first item in the current node

Print “1) {second item in the current node}”

Print “2) {fourth item in the current node}”

Create a variable called “answer” which gets the response to “Type 1 or 2: “

If the answer is in (“1”, “2”):

Give keepGoing False

Else:

Print “Incorrect input, please type either a ‘1’ or a ‘2’”

Return answer

* **getDefaultGame()**
  + creates a single-node default game
  + returns that data structure

Make a dictionary called “gameFile”

gameFile = {

“start”: (“Welcome to the game! What would you like to do?”, “Start over”, “start”, “Quit”, “quit”)

}

Return gameFile

* **editNode()**
  + given the current game structure...
  + list all the current node content  (json.dumps() would be an easy way to do this)
  + get a node name
  + if that node exists
    - copy that node to newNode
  + otherwise...
    - create newNode with empty data
  + use editField() to allow user to edit each node
  + return the now edited newNode

print(gameFile)

the variable {node} gets the user’s response to (“Here is your current game! What node would you like to edit/create? “)

if {node} is equal to an existing node:

playerNodeList = what the existing node contains

Otherwise:

playerNodeList = [“nothing”, “nothing”, “nothing”, “nothing”, “nothing”, “nothing”]

listPart = [“node name”, “situation”, “option 1”, “option 1 pathway”, “option 2”, “option 2 pathway”]

call editField() with the arguments playerNodeList[0] and listPart[0]

call editField() with the arguments playerNodeList[1] and listPart[1]

call editField() with the arguments playerNodeList[2] and listPart[2]

call editField() with the arguments playerNodeList[3] and listPart[3]

call editField() with the arguments playerNodeList[4] and listPart[4]

call editField() with the arguments playerNodeList[5] and listPart[5]

make a new node in the dictionary using the playerNodeList

*# use dictionary[playerNodeList[0]] = playerNodeList[1], playerNodeList[2], etc*

* **editField()**
  + get a field name
  + print the field's current value
  + if the user presses 'enter' immediately
    - retain the current value
  + otherwise...
    - use the new value

give editField() the parameter nodePart and nodePartName and number

nodePart gets the input from “Your current nodePartName is “nodePart.” What would you like it to be? “

return nodePart[number]

* **saveGame()**
  + save the game to a data file
  + you can preset the file name (eg 'game.dat')
  + print the current game dictionary in human-readable format
  + Save the file in JSON format

Make a file pointer called saveFile which gets open(“gameFile.json”, “w”)

Use json.dump(gameFile, saveFile, indent=2)

Close the outFile

Print(“Gotcha! I saved your game’s data to gameFile.json! Here it is if you want to look at it.”

Print(gameFile)

* **loadGame()**
  + presume there is a data file named 'game.dat' in the current directory
  + open that file
  + load the data into the game object
  + return that game object

make a file pointer called loadFile which gets open(“gameFile.json”, “r”)

the variable {gameFile} gets json.load(loadFile)

close inFile