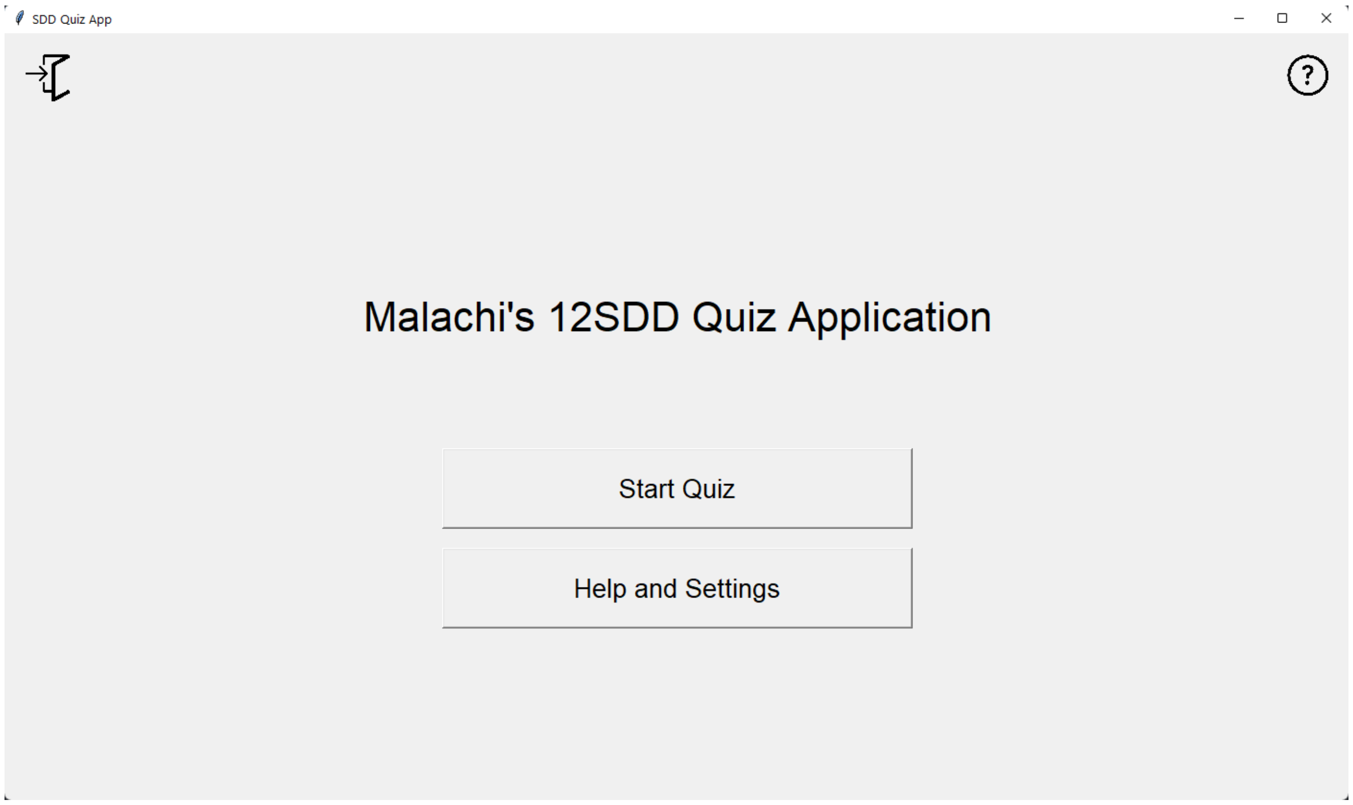
12 SDD AT1 Project Documentation

Malachi English



Contents

Stage 1: Defining and Understanding the problem

Target Audience

The target audience of this app are students studying HSC Software Design and Development.

Design Specifications

Design Specification	User's perspective
Inclusive instructions/help menu with feedback/messages	- Feedback that is easy to understand - Language that is not discriminatory or offensive - Application is clearly explained with instructions
At least 10 questions with answers and quiz scores.	- Questions with a way to submit answers - Clear feedback on the user's score, including how many questions they got correct, and what questions they got correct
Effective, consistent screen design	- Substantial white space between elements - Consistent font, font size, colors, Gui elements - Purpose of buttons is clear
4 screen elements, 3 navigation elements	<u>4 Screen elements</u> - Radio buttons - Check buttons - Scroll bar - 2 images <u>3 Navigation elements</u> - Menu page - Buttons - Appropriate space between elemetns

Stage 2: Planning and Designing the solution

System Management

Gantt Chart

	27/10/2022	03/11/2022	10/11/2022	28/11/2022	04/12/2022	05/12/2022
defining and understanding the problem		🟩				
Define target audience		🟩				
Design specifications		🟩				
planning and designing a solution			🟩			
Set up Logbook			🟩			
Partial Data Dictionary			🟩			
Storyboard			🟩			
IPO Diagram			🟩			
Context Diagram			🟩			
implementing the solution				🟩		
User Documentation				🟩		
Create the App				🟩		
testing and evaluating the solution					🟩	
Testing with Users					🟩	
Evaluating					🟩	
maintenance (future considerations)						🟥 milestone

Logbook

Dates 10/11/22
Time Taken 120 minutes
Progress/Issues - Created the layout for the questions page. (use of widgets and grid - code adaption from exercise 1.2.15) - Set up the project with different classes - Radio-buttons weren't appearing as buttons, but as text with checkable circles.

- Researched the use of classes in tkinter as pages

Issue Resolution
- Had to use ‘Radiobutton’ instead of ‘ttk.Radiobutton’ and edit the style of the Radiobutton

References
- https://python-course.eu/tkinter/radio-buttons-in-tkinter.php
- https://www.digitalocean.com/community/tutorials/tkinter-working-with-classes

Dates
11-13/11/22

Time Taken
200 minutes

Progress/Issues
- Created functionality to switch pages
- Added a menu screen
- Took a long time to get the previous page to clear, and the new frame to appear in the correct place

Issue Resolution
- Created one frame in the app that would hold all the different pages. A variable called App.currentpage would track the currently active page, and to switch pages, the program would forget the current page, and pack the new one

References
- https://www.geeksforgeeks.org/tkinter-application-to-switch-between-different-page-frames/
- https://stackoverflow.com/questions/12364981/how-to-delete-tkinter-widgets-from-a-window

Dates
14-15/11/22

Time Taken
140 minutes

Progress/Issues
- Made a data file so the question pages could be dynamically created for each question
- Put in a progress bar for the quiz

Issue Resolution
-

References
- https://www.geeksforgeeks.org/python-tkinter-scrollbar/

Dates
16-20/11/22

Time Taken
270 minutes

Progress/Issues
- Created a results page
- The scrollable region took a while to get working with the scrollbar, and even longer with the mousewheel as well

Issue Resolution
- Had to put the frame in a canvas, and configure the canvas correctly with the scrollbar. To get the mouse-wheel input working, I had to bind it to a scrolling function.

References
- https://stackoverflow.com/questions/16188420/tkinter-scrollbar-for-frame
- https://stackoverflow.com/questions/17355902/tkinter-binding-mousewheel-to-scrollbar

Dates
21-23/11/22

Time Taken
160 minutes

Progress/Issues
- Created two icon buttons for ‘exit’ and ‘help’. Images initially didn’t show correctly on the buttons (code adaption from exercise 2.2.9 to display images)
- Added functionality for the ‘exit’ button, including message prompts (code adaption from exercise 2.2.9 to trigger a messagebox)
- Added a help window (code adaption from exercise 2.2.9 to open a new window)

Issue Resolution
- Button was not being packed correctly

References
- https://www.geeksforgeeks.org/python-add-image-on-a-tkinter-button/

Dates
24-25/11/22

Time Taken
140 minutes

Progress/Issues
- Filled in all the questions for the quiz
- As the questions/answers were longer than the initial placeholder text, output was sometimes weird

Issue Resolution
- Had to set the width of the buttons explicitly

- Had to set the wrap length of various buttons and labels (including in the results section)
References - https://stackoverflow.com/questions/11949391/how-do-i-use-tkinter-to-create-line-wrapped-text-that-fills-the-width-of-the-win

Dates 26-27/11/22
Time Taken 260 minutes
Progress/Issues - Filled in the 'help and settings' window - Added accessibility options. The 'larger text size' option messed up the output in some cases - Checkboxes would not be checked when reopening the window
Issue Resolution - Had to try different values for widgets affected by the larger text size so that they fit correctly - Created variables in the app class to track checked variables, so that they can be already checked when the window opens
References - https://stackoverflow.com/questions/37595078/tkinter-is-there-a-way-to-check-checkboxes-by-default - https://stackoverflow.com/questions/19163658/tkinter-checkbutton-and-event-callback-function

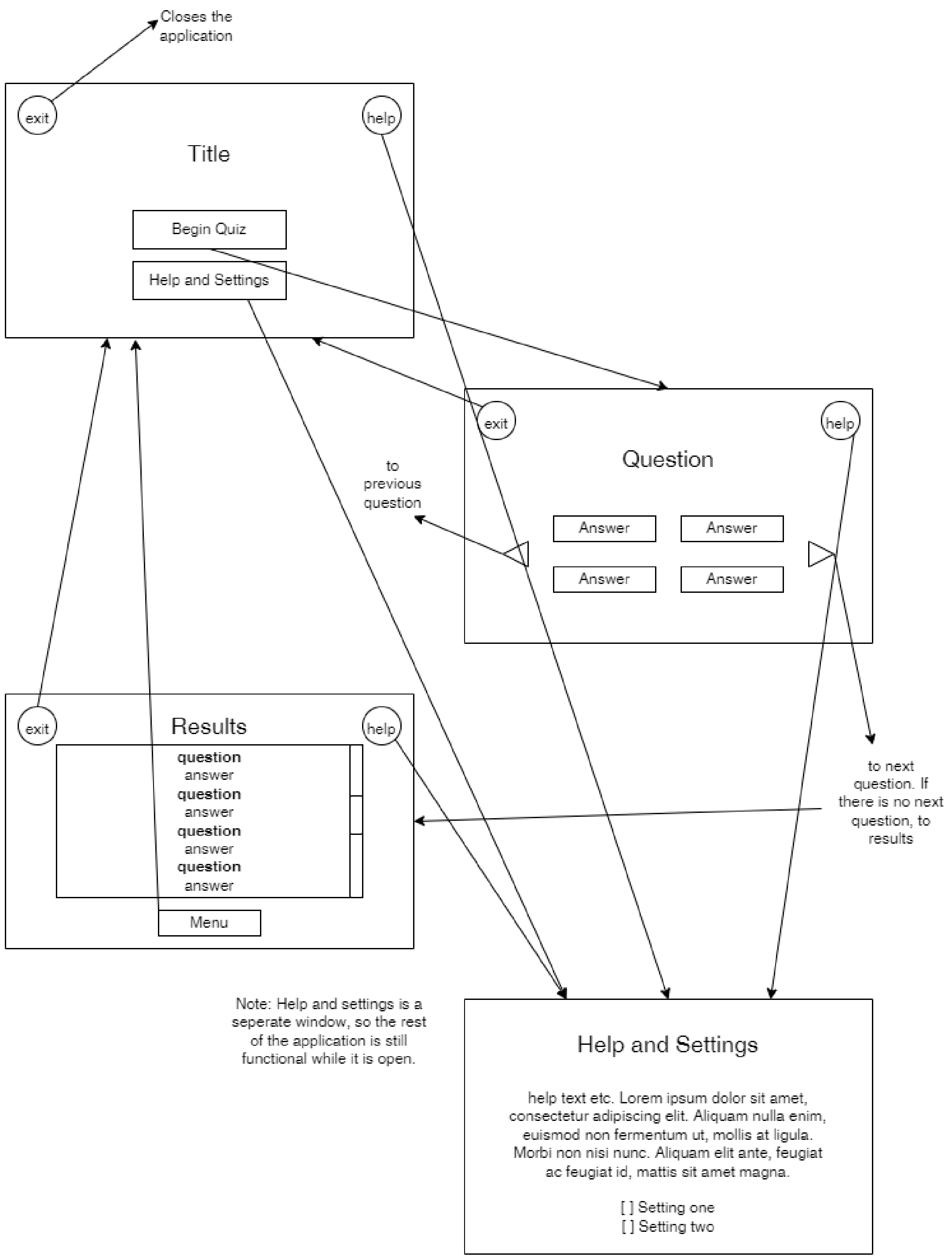
Dates 28/11/22-3/12/22
Time Taken 150 minutes
Progress/Issues - Polishing - Proofreading - Tidying up the code
Issue Resolution -
References -

System Modelling

Partial Data Dictionary

Data Item	Data Type	Description
QuestionPage.questiondata	Dictionary with string pairs	Contains all the answers and whether they are correct or incorrect for each question
QuestionPage.question_bank	List of lists, containing a string and another list	Contains all the questions with their answers
QuestionPage.answerorder	List	The numbers 0-3, depicting the order in which answers will appear, after the list is shuffled
App.location	string	Tracks the screen the user is currently on (e.g. menu or results)
App.correctanswers	integer	Tracks the amount of correct answers the user has entered
App.answersheet	List of strings	Tracks the order or correct and incorrect answers
App.currentquestion	integer	Tracks what question the user is up to
Questionpage.row	integer	Used for the construction of the answer grid
Quetionpage.column	integer	Used for the construction of the answer grid
ResultsPage.rowcounter	integer	Used for the construction of the results section

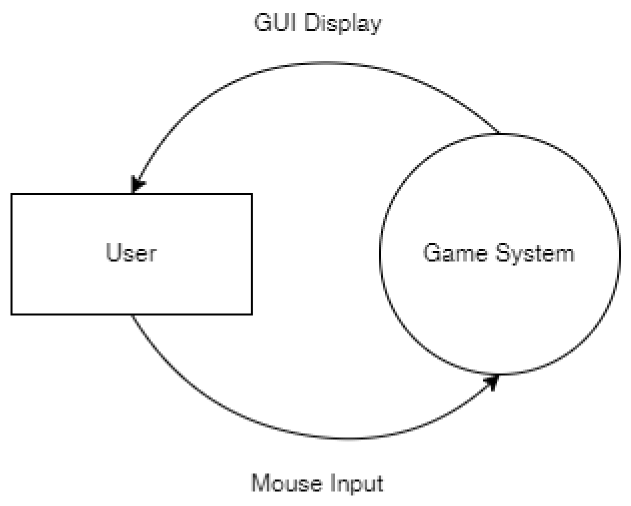
Storyboard or sample screen layouts


Closes the application
exit
help
Title
Begin Quiz
Help and Settings
exit
help
Question
to previous question
Answer
Answer
Answer
Answer
exit
Results
help
question
answer
question
answer
question
answer
question
answer
Menu
to next question. If there is no next question, to results
Note: Help and settings is a seperate window, so the rest of the application is still functional while it is open.
Help and Settings
help text etc. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam nulla enim, euismod non fermentum ut, mollis at ligula. Morbi non nisi nunc. Aliquam elit ante, feugiat ac feugiat id, mattis sit amet magna.
[ ] Setting one
[ ] Setting two

IPO diagram

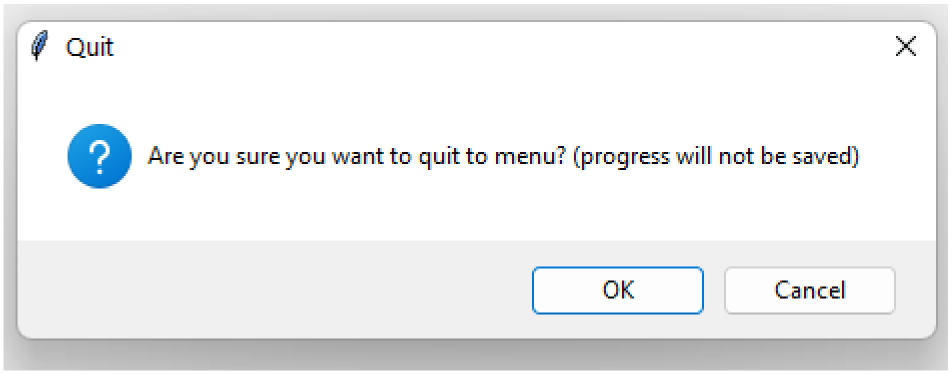
Inputs	Processes	Outputs
- Question answers - Navigational Instructions - Accessibility settings	- Randomising answer order - Tracking correct answers - Tracking question number - Finding the next page to display - Calculating score - Calculating progressbar value - Checking accessibility settings - Providing appropriate responses based on location in the app	- Questions - Answer options - Quiz results, with accessibility settings if checked - Navigational buttons (some with icons) - Help menu - App menu and title - prompts

Context diagram

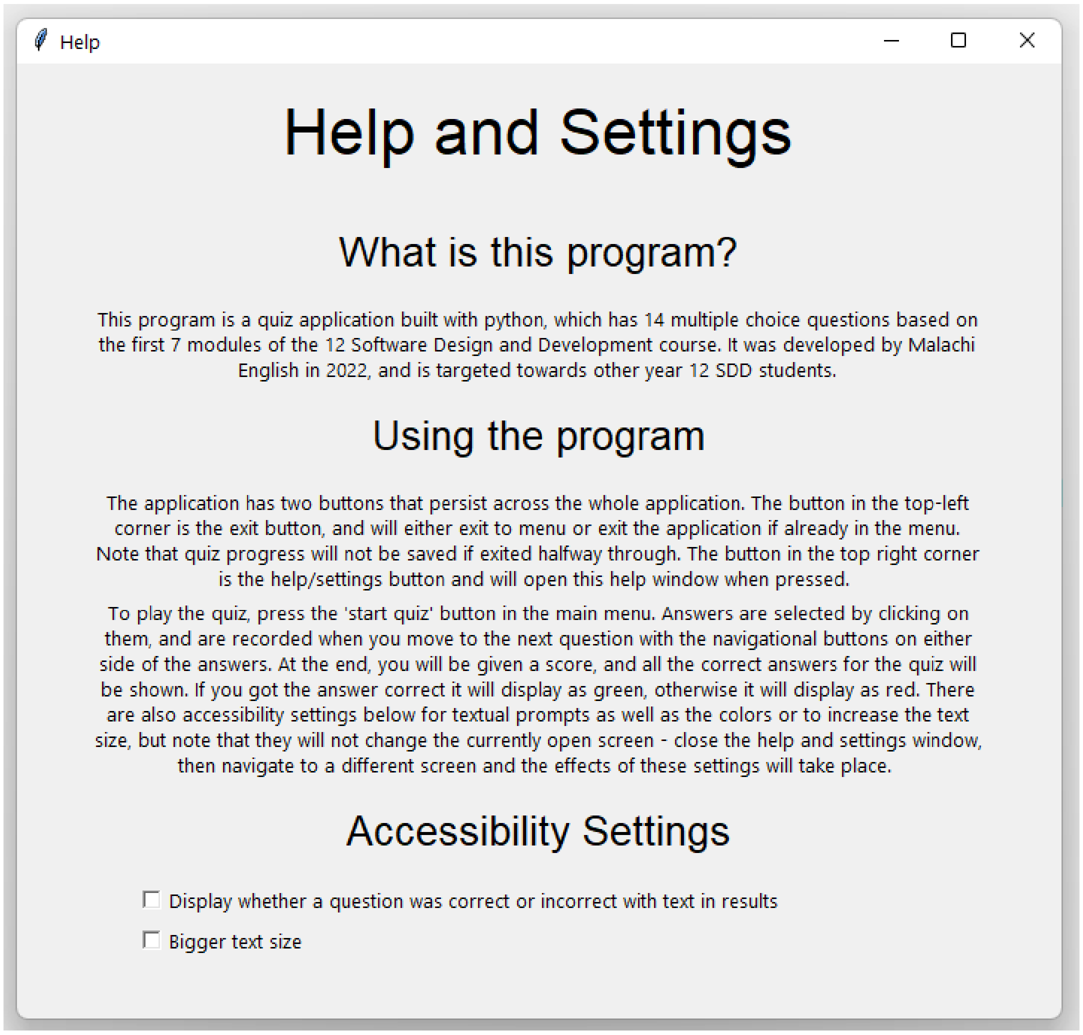


Stage 3: Implementation of the solution

User Documentation


Quit

Are you sure you want to quit to menu? (progress will not be saved)


Help

Help and Settings

What is this program?

This program is a quiz application built with python, which has 14 multiple choice questions based on the first 7 modules of the 12 Software Design and Development course. It was developed by Malachi English in 2022, and is targeted towards other year 12 SDD students.

Using the program

The application has two buttons that persist across the whole application. The button in the top-left corner is the exit button, and will either exit to menu or exit the application if already in the menu. Note that quiz progress will not be saved if exited halfway through. The button in the top right corner is the help/settings button and will open this help window when pressed.

To play the quiz, press the 'start quiz' button in the main menu. Answers are selected by clicking on them, and are recorded when you move to the next question with the navigational buttons on either side of the answers. At the end, you will be given a score, and all the correct answers for the quiz will be shown. If you got the answer correct it will display as green, otherwise it will display as red. There are also accessibility settings below for textual prompts as well as the colors or to increase the text size, but note that they will not change the currently open screen - close the help and settings window, then navigate to a different screen and the effects of these settings will take place.

Accessibility Settings

☐ Display whether a question was correct or incorrect with text in results

☐ Bigger text size

Internal and Intrinsic Documentation

```
# Created by Malachi English 3/12/22
# 12SDD AT1 - SDD quiz app
# Application with a multiple choice quiz, a results page, a menu page, and a
help and settings page.



# Module imports
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from tkinter import font
import json
import random



class QuestionPage(Frame):
    def __init__(self, master, pagenumber):
        Frame.__init__(self)

        # Opens file and loads data
        file = open("./questions.json")
        self.question_bank = json.load(file)
        self.questiondata = self.question_bank[pagenumber]

        # Gets each answer from the datafile and connects them to a 'correct'
or 'incorrect' value
        self.questiongroup = {
            self.questiondata[1][0]: "Correct",
            self.questiondata[1][1]: "Incorrect0",
            self.questiondata[1][2]: "Incorrect1",
            self.questiondata[1][3]: "Incorrect2"
        }

        file.close()
        # Creates the 'masterframe' - the main frame that everything in this
class will go in.
        self.masterframe = Frame(self)
        self.masterframe.pack(anchor=CENTER, expand=True)

        # Divides the screen up into various frames (left, question, right,
bottom), to organise widget placement
        self.leftframe = Frame(self.masterframe)
        self.leftframe.grid(row=1, column=0, sticky="nsew")

        self.questionframe = Frame(self.masterframe, height=200)
        self.questionframe.grid(row=0, column=1, sticky="nsew")

        self.answerframe = Frame(self.masterframe, height=400, width=100)
        self.answerframe.grid(row=1, column=1, sticky="nsew")

        self.rightframe = Frame(self.masterframe)
        self.rightframe.grid(row=1, column=2, sticky="nsew")

        self.bottomframe = Frame(self.masterframe)
        self.bottomframe.grid(row=2, column=1, sticky="nsew")

        # Displays the question from the datafile
        self.question = Label(self.questionframe, text=self.questiondata[0],
font=("Ariel", 22), pady=2, wraplength=600, height=5, justify="center")
        self.question.pack()

        # Creates the StringVar for the multiple choice answers
        self.radiogroup1 = StringVar()

        # Used for construction of the 2x2 answer grid
        self.row=1
        self.column=0

        # Shuffles the answers so they appear in a random order
        self.answerorder = [0, 1, 2, 3]
        random.shuffle(self.answerorder)

        if App.biggertext:
            self.answerfont = ('Ariel', 14)
        else:
            self.answerfont = ('Ariel', 10)

        # Creates a 2x2 grid of toggle buttons in the randomised order
previously defined.
        for i in self.answerorder:
            Radiobutton(
                self.answerframe,
                indicatoron=False,
                text=list(self.questiongroup)[i],
                font=self.answerfont,
                variable=self.radiogroup1,
                value=list(self.questiongroup.values())[i],
                width=35,
                height=3,
                pady=2,
                wraplength=275,
                justify="center"
                ).grid(row=self.row, column=self.column, padx=20, pady=10)

            # Moves the row/column of the next radiobutton so that they appear
in a 2x2 grid
            if self.column == 0: self.column = 1;
            elif self.column == 1: self.column = 0; self.row = 2


        # Button that goes to the previous question
        self.leftbutton = Button(self.leftframe, text=" < ", command=lambda :
App.previous(App, master), padx=10, pady=10)
        self.leftbutton.pack(padx=70, pady=70, expand=True)

        # Button that goes to the next question
        self.rightbutton = Button(self.rightframe, text=" > ", command=lambda :
App.next(App, master), padx=10, pady=10)
        self.rightbutton.pack(padx=70, pady=70, expand=True)

        # Progress bar that shows progress through quiz
        self.progressbar = ttk.Progressbar(self.bottomframe,
orient='horizontal', length=500, mode='determinate')
        self.progressbar.pack(pady=35)
        self.progressbar['value'] = 1 / len(self.question_bank) * 100
```

```
class ResultsPage(Frame):
    def __init__(self, master):
        Frame.__init__(self)

        # Sets the location to results
        App.location = "results"

        # Opens and loads the data from the question file
        file = open("./questions.json")
        self.question_bank = json.load(file)
        file.close()

        # Creates the 'masterframe' - the main frame that everything in this
class will go in.
        self.masterframe = Frame(self)
        self.masterframe.pack(anchor=CENTER, expand=True)

        # Displays title of the page (score)
        self.scoretitle = Label(self.masterframe, text="Results",
font=("Ariel", 32), pady=25)
        self.scoretitle.grid(row=0, column=0)

        # Displays title of the page (score)
        self.scoretitle = Label(self.masterframe, text=f"Your Score:
{App.correctanswers}/{len(self.question_bank)}", font=("Ariel", 22), pady=25)
        self.scoretitle.grid(row=1, column=0)

        # Creates a canvas where the scrollable content will be lcoated
        self.scrollbox = Canvas(self.masterframe, height=400, width=800)
        self.scrollbox.configure(scrollregion=self.scrollbox.bbox("all"))
        self.scrollbox.grid(row=2, column=0, sticky = 'ew')

        # Frame for all the results
        self.resultsframe = Frame(self.scrollbox)

        # Scrollbar to scroll through results
        self.scrollbar = Scrollbar(self.masterframe, orient="vertical",
command=self.scrollbox.yview)
        self.scrollbar.configure(scrollregion=self.scrollbox.bbox("all"))
        self.scrollbar.grid(row=2, column=1, sticky="ns")

        # Defines the scroll region of scrollbox
        self.resultsframe.bind_all(
            "<Configure>",
            lambda event: self.scrollbox.configure(
                scrollregion=self.scrollbox.bbox("all")
            )
        )

        # Binds the mousewheel to the mousewheel function (enables scrolling)
        self.resultsframe.bind_all(
            "<MouseWheel>",
            self.MouseWheel
        )

        # Creates a window in the canvas with the results inside of it
        self.scrollbox.create_window((400, 0), window=self.resultsframe,
anchor="n")

        # Updates the scrollbar when the canvas is scrolled
        self.scrollbox.configure(yscrollcommand=self.scrollbar.set)

        # Sets the fontsize based on accessibility settings
        if App.biggertext:
            self.answerfont = ("Ariel", 14)
            self.questionfont = ("Ariel", 18)
        else:
            self.answerfont = ("Ariel", 10)
            self.questionfont = ("Ariel", 16)

        # Used for construction of the question/answer section
        self.rowcounter = 0
        self.answercounter = 0

        # Displays all the questions and correct answers as Labels
        for i in self.question_bank:
            # Question
            Label(self.resultsframe, text=f"{i[0]}", font=self.questionfont,
padx=340, wraplength=600, justify="center").grid(row=(self.rowcounter + 1),
column=0)
```

```
            self.rowcounter += 1

            # If answer was correct, displays as green
            # If App.showcorrect is true, also displays the text 'You got it 
correct'
            if App.answersheet[self.answercounter] == "correct":
                Label(
                    self.resultsframe,
                    text=f"{i[1][0] + ('     (You got it correct)' if 
App.showcorrect == True else '')}",
                    fg='green',
                    font=self.answerfont,
                    wraplength=600
                ).grid(row=(self.rowcounter+1), column=0)

            # If answer was incorrect, displays as red
            # If App.showcorrect is true, also displays the text 'You got it 
incorrect'
            else:
                Label(
                    self.resultsframe,
                    text=f"{i[1][0] + ('     (You got it incorrect)' if 
App.showcorrect == True else '')}",
                    fg='red',
                    font=self.answerfont,
                    wraplength=600
                ).grid(row=(self.rowcounter+1), column=0)
            self.rowcounter += 1
            self.answercounter += 1

        # Button to return to menu
        self.returnbutton = Button(self.masterframe, text="Return To Menu", 
pady=2, padx=2, font=("Ariel", 14), command = lambda : App.menu(App, master))
        self.returnbutton.grid(row=3, column=0, pady=50)


    def MouseWheel(self, event):
        # scrolls scrollbox when mousewheel is used
        # event.delta is divided by 120 and multiplied by -1 so that it scrolls 
in the correct direction and does not scroll too fast
        self.scrollbox.yview("scroll", int(-1*(event.delta/120)),"units")
```

```
class StartPage(Frame):
    def __init__(self, master):
        Frame.__init__(self)

        # Creates the 'masterframe' - the main frame that everything in this
class will go in.
        self.masterframe = Frame(self)
        self.masterframe.pack(anchor=CENTER, expand=True)

        # Title
        self.title = Label(self.masterframe, pady=100, font=("Ariel", 32),
text="Malachi's 12SDD Quiz Application")
        self.title.pack()

        self.buttonfont = font.Font(family='Helvetica', size=20)

        # Button to begin the quiz
        self.startbutton = Button(self.masterframe, text="Start Quiz",
width=30, pady=15, font=self.buttonfont, command = lambda : App.beginquiz(App,
master))
        self.startbutton.pack(pady=10)

        # Button to launch help and settings window
        self.helpbutton = Button(self.masterframe, text="Help and Settings",
width=30, pady=15, font=self.buttonfont, command = lambda : App.help(App,
master))
        self.helpbutton.pack(pady=10)



class HelpPage(Frame):
    def __init__(self, master):
        Frame.__init__(self)
        self.headingfont = ("Ariel", 20)

        # Creates the 'masterframe' - the main frame that everything in this
class will go in.
        self.masterframe = Frame(master)
        self.masterframe.pack(anchor=CENTER, expand=True)

        # Title
        self.title = Label(self.masterframe, font=("Ariel", 32), pady=20,
text="Help and Settings")
        self.title.grid(row=0, column=0)

        # What is this program?
        self.whatheading = Label(self.masterframe, font=self.headingfont,
pady=15, text="What is this program?")
        self.whatheading.grid(row=1, column=0)

        self.whatbody = Label(self.masterframe, wraplength=600,
justify="center", text="This program is a quiz application built with python,
which has 14 multiple choice questions based on the first 7 modules of the 12
Software Design and Development course. It was developed by Malachi English in
2022, and is targeted towards other year 12 SDD students.")
        self.whatbody.grid(row=2, column=0)

        # Using the program
        self.usingheading = Label(self.masterframe, font=self.headingfont,
pady=15, text="Using the program")
        self.usingheading.grid(row=3, column=0)

        self.usingbody1 = Label(self.masterframe, wraplength=600,
justify="center", text="The application has two buttons that persist across the
whole application. The button in the top-left corner is the exit button, and
will either exit to menu or exit the application if already in the menu. Note
that quiz progress will not be saved if exited halfway through. The button in
the top right corner is the help/settings button and will open this help window
when pressed. ")
        self.usingbody1.grid(row=4, column=0)

        self.usingbody2 = Label(self.masterframe, wraplength=600,
justify="center", text="To play the quiz, press the 'start quiz' button in the
main menu. Answers are selected by clicking on them, and are recorded when you
move to the next question with the navigational buttons on either side of the
answers. At the end, you will be given a score, and all the correct answers for
the quiz will be shown. If you got the answer correct it will display as green,
otherwise it will display as red. There are also accessibility settings below
for textual prompts as well as the colors or to increase the text size, but
note that they will not change the currently open screen - close the help and
settings window, then navigate to a different screen and the effects of these
settings will take place. ")
        self.usingbody2.grid(row=5, column=0)

        # Accessibility settings
        self.accessibilityheading = Label(self.masterframe,
font=self.headingfont, pady=15, text="Accessibility Settings")
        self.accessibilityheading.grid(row=6, column=0)

        # Show correct/incorrect next to results
        self.showcorrect = BooleanVar()
        self.showcorrectcheck = Checkbutton(
            self.masterframe,
            text="Display whether a question was correct or incorrect with text
in results",
            variable=self.showcorrect,
            onvalue=True,
            offvalue=False,
            command=self.checkchanged
        )
        # align it to the left
        self.showcorrectcheck.grid(row=7, column=0, sticky='w', padx=30)

        # If it has been set to true, check it when the window is opened.
        if App.showcorrect:
            self.showcorrectcheck.select()

        # Bigger text size
        self.biggertext = BooleanVar()
        self.biggertextcheck = Checkbutton(
            self.masterframe,
            text="Bigger text size",
            variable=self.biggertext,
            onvalue=True,
            offvalue=False,
            command=self.checkchanged
        )
        # align it to the left
        self.biggertextcheck.grid(row=8, column=0, sticky='w', padx=30)
        # If it has been set to true, check it when the window is opened.
        if App.biggertext:
            self.biggertextcheck.select()


    def checkchanged(self):
        # Tracks if showcorrect has been set to true or false
        App.showcorrect = self.showcorrect.get()
        # Tracks of biggertext has been set to true or false
        App.biggertext = self.biggertext.get()



class App():
    # The App class is responsible for what appears on the screen, so anything
in the App class is what is shown.
    def __init__(self, master):
        self.style = ttk.Style(master)
        self.style.theme_use('default')
        # Makes it so the radiobutton has no indicator (e.g. filled/unfilled
circle) next to it
        self.style.configure('TRadiobutton', indicatoron=0)

        # Initialise variables
        App.correctanswers = 0
        App.location = "start"
        App.showcorrect = False
        App.biggertext = False

        # Almost everything in the app appears in this frame
        App.container = Frame(master)
        App.container.pack(side=TOP, anchor='center')

        # Defines the help icon and makes it smaller
        App.helpicon = PhotoImage(file='./helpicon.png')
        App.helpicon = App.helpicon.subsample(12, 12)

        # Defines the quit icon and makes it smaller
        App.quiticon = PhotoImage(file='./quiticon.png')
        App.quiticon = App.quiticon.subsample(20, 20)
        # Help button, which always appears in the top-right corner
        self.help_button = Button(master, image=App.helpicon, borderwidth=0,
command = lambda : App.help(App, master))
        self.help_button.pack(side=RIGHT, anchor=NE, pady=20, padx=20,
expand=False)

        # Quit button, which always appears in the top-left corner
        self.quit_button = Button(master, image=App.quiticon, borderwidth=0,
command = lambda : App.quit(App, master))
        self.quit_button.pack(side=LEFT, anchor=NW, pady=20, padx=20)

        self.menu(master)


    def menu(self, master):
        # Initialises variables
        App.location = "start"
        App.correctanswers = 0
        App.currentquestion = 0
        App.answersheet = []

        # If there is already a currentpage, clear it.
        try: App.currentpage.forget()
        except AttributeError: pass

        # Packs the menu page (start page)
        App.currentpage = StartPage(App.container)
        App.currentpage.pack(anchor=CENTER, expand=True)


    def help(self, master):

        try:
            # If user has already opened a help window, lift it to the top
instead of opening a new one.
            if self.helpwindow.state() == 'normal': self.helpwindow.lift()
        except:
            # Create a new window with dimensions 700x640 and title 'help'
            self.helpwindow = Toplevel(master)
            self.helpwindow.geometry("700x640")
            self.helpwindow.title("Help")
            # Creates a frame inside the window
            self.helpframe = Frame(self.helpwindow)
            self.helpframe.pack()

            # Packs the help page in the helpframe
            self.helpp = HelpPage(self.helpframe)
            self.helpp.pack()


    def beginquiz(self, master):
        # Sets the location to quiz
        App.location = "quiz"
        # Clears the currentpage frame
        App.currentpage.forget()

        # Sets the currentquestion to the first one.
        App.currentquestion = 0

        # Displays the question/answers on the screen
        App.currentpage = QuestionPage(App.container, self.currentquestion)
        App.currentpage.pack(anchor=CENTER, expand=True)


    def next(self, master):
        # Records if user got the question correct
        if App.currentpage.radiogroup1.get() == "Correct":
            App.correctanswers += 1
            App.answersheet.append("correct")
        else:
            App.answersheet.append("incorrect")

        # Clears the currentpage frame
        App.currentpage.forget()

        # Moves to the next question
        App.currentquestion += 1

        try:
            # If there is a next question, displays it on the screen
            App.currentpage = QuestionPage(App.container, App.currentquestion)
            App.currentpage.progressbar['value'] = (App.currentquestion+1) / 
len(App.currentpage.question_bank) * 100
        except IndexError:
            # Otherwise, displays the results of the quiz
            App.currentpage = ResultsPage(App.container)
        App.currentpage.pack(anchor=CENTER, expand=True)


    def previous(self, master):
        # Function to navigate to the previous question

        # Makes sure user is not at the first question
        if App.currentquestion > 0:
            # clears the currentpage frame
            App.currentpage.forget()
            # Moves to the previous question
            App.currentquestion -= 1
            # Resets progress made by one question
            App.correctanswers -= 1
            App.answersheet.pop()

            # Displays the question/answers on the screen
            App.currentpage = QuestionPage(App.container, App.currentquestion)
            App.currentpage.pack(anchor=CENTER, expand=True)
            App.currentpage.progressbar['value'] = (App.currentquestion+1) / 
len(App.currentpage.question_bank) * 100


    def quit(self, master):
        # If user is in the menu screen, quit will ask to exit the program
        if App.location == "start":
            if messagebox.askokcancel(title="Quit", message="Are you sure you 
want to quit?"):
                master.destroy()
        # If user is in the quiz, quit will ask to exit to menu, warning that 
progress will not be saved.
        elif App.location == "quiz":
            if messagebox.askokcancel(title="Quit", message="Are you sure you 
want to quit to menu? (progress will not be saved)"):
                App.menu(App, master)
        # If user is in the results, quit will ask to exit to menu
        elif App.location == "results":
            if messagebox.askokcancel(title="Quit", message="Quit to menu?"):
                App.menu(App, master)



def main():
    # Creates the main window with title 'SDD Quiz App' and dimensions
1400x800.
    root = Tk()
    root.title("SDD Quiz App")
    root.geometry("1400x800")
    # minimum size the window can be shrunk to
    root.minsize(1400, 700)

    app = App(root)
    #app.pack(side="top", fill="both", expand=True)
    root.mainloop()



if __name__ == "__main__": main()
```

Stage 4: Testing and Evaluating the solution

Testing Process

User’s IT experience	PC/Mac	Observations of User	User’s feedback
Medium-high	PC	User had no troubles navigating or undestanding the application.	I liked that it had a progress bar at the bottom and that it had a help menu at the top right corner
low-medium	PC	User was slightly confused with the results page. I afterwards added a short hint explaining what	I Initially thought the answers in the results section were the answers I submitted, so was confused a bit by that. Otherwise the application was fine
low-medium	PC	User did not completely understand the application immediately, but was able to work it out.	I loved the inclusion of accessibility options, although I didn’t realise you had to close the window for them to work.

Evaluation Process (2 sentences each)

Verification- how did the app perform
I think the App performed well, it served the purpose of a quiz application, with questions, results and various other functions to imporve the quality-of-life. User testing revealed a few clarity issues, but the application was updated in an attempt to fix these. Response times were not necissarily a problem, although there is a visible flicker between pages in the app, which is irritating, but hard to fix with the current implementation of the solution, and the limitations of python and tkinter. Overall though, the application performed fine.

Validation - compare your final app to the 4 original design specifications
The App complied with all of the 4 original deign specifications - it had inclusive instructions/help menu with feedback/messages, at least 10 questions with answers and quiz scores, effective consistent screen design, 4 screen elements, and 3 navigation elements. From the User’s perspective in the design specifications table, all of the dot-points were also followed.

Stage 5: Maintenance of the solution

Maintenance - Discuss the solution’s future maintenance, issues, or possible updates (2 sentences)

Both python and the Tkinter package are subject to change in the future, so the solution will need to be updated to accommodate for any of these changes. Possible future updates to improve the solution include different colour themes (e.g. light and dark), or an option to upload your own quiz to the application.