```
import pygame
from random import randint, choice
import time
from crossword_generator import main

#RGB values
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
GREY = (220,220,220)
BLUE = (30, 144, 255)

SCREEN_SIZE_X = 1350
SCREEN_SIZE_Y = 810
FPS = 60

pygame.init()

fps_clock = pygame.time.Clock()

screen = pygame.display.set_mode([SCREEN_SIZE_X, SCREEN_SIZE_Y])
pygame.display.set_caption("Crossoword Generator")

#loads the images to be used
button_sprite_1 = pygame.image.load("button_4.png")
button_sprite_2 = pygame.image.load("button_8.png")
button_sprite_3 = pygame.image.load("button_9.png")


#A class which handles the list of clues displayed to the right of the crossword.

class ClueList():

    def __init__(self, nodes):

        self.__nodes = nodes

        #Text() objects for the "across" and "down" headings of the clue list.
        self.__across_title = Text("chalkboard", 35, BLACK, [SCREEN_SIZE_X/1.68, SCREEN_SIZE_Y/5.5], "Across", Fals
        self.__down_title = Text("chalkboard", 35, BLACK, [SCREEN_SIZE_X/1.26, SCREEN_SIZE_Y/5.5], "Down", False, F

        #A list of Text() objects with which a for loop goes through and updates each one.
        self.__text_to_update = [self.__across_title, self.__down_title]

        #These line space variables are used for the for loop and
        #ensure that each word in the clue list has an appropiate vertical space between them.
        line_space_across = self.__across_title.getTextRect()[3]
        line_space_down = self.__down_title.getTextRect()[3]

        for node in self.__nodes:

            #A string in the format of "number. word" e.g. "1. pomme".
            text_string = f"{node.grid_square_number}. {node.getClueWord()}"

            #If the word is an adjective then it needs to have its gender shown in brackets.
            if node.getGender() != "":
                text_string += f" ({node.getGender()})"

            if node.getDirection() == "across":
                text = Text("chalkboard", 20, BLACK, [self.__across_title.getCoords()[0], self.__across_title.getCo
                line_space_across += text.getTextRect()[3]

            else:
                text = Text("chalkboard", 20, BLACK, [self.__down_title.getCoords()[0], self.__down_title.getCoords
                line_space_down += text.getTextRect()[3]

            self.__text_to_update.append(text)

    def update(self):

        for text in self.__text_to_update:
            text.update()


#A class which creates objects for each square in the Crossword().
```

```
class GridSquare():

    def __init__(self, rect_values, letter_answer):

        #A list in the format (x_pos, y_pos, width, height) which contains the neccessary details to draw a rectang
        self.__rect_values = rect_values

        #Stores the letter which this grid square should contain.
        self.__letter_answer = letter_answer

        #Stores the letter which the user has entered.
        self.__letter_entered = ""

        #The little word-line number which may be displayed in the top left-hand corner if this is the first square
        self.__number = 0

        #A letter_answer of "X" indicates that this is a blank square which should be filled with black.
        if self.__letter_answer == "X":
            self.__blank = True
        else:
            self.center_x = (self.__rect_values[0] + self.__rect_values[2] / 2)
            self.center_y = (self.__rect_values[1] + self.__rect_values[3] / 2)
            self.__letter_object = ""
            self.__number_object = ""
            self.__blank = False

        #A list in the format [(start_x, end_x), (start_y, end_y)] which shows the range of pixels which this recta
        self.__hitbox = [(self.__rect_values[0], self.__rect_values[0] + self.__rect_values[2]), (self.__rect_value

        #A boolean to show whether the user has clicked on this square to type into it.
        self.__activated = False

        #If the letter in this square is the correct letter.
        self.__correct = True

    def getActivated(self):
        return self.__activated

    def getLetterAnswer(self):
        return self.__letter_answer

    def getLetterEntered(self):
        return self.__letter_entered

    def getNumber(self):
        return self.__number

    def getBlank(self):
        return self.__blank

    def setCorrect(self, bool):
        self.__correct = bool

    def setActivated(self, bool):
        self.__activated = bool

    def addNumber(self, number):

        self.__number = number
        self.__number_object = Text("chalkboard", 15, BLUE, [self.__rect_values[0]+4, self.__rect_values[1]+3], str

    def keyDown(self, letter):

        if self.__activated:
            self.__letter_entered = letter
            self.__letter_object = Text("chalkboard", 25, BLACK, [self.center_x, self.center_y], self.__letter_ente

    def mouseClick(self):

        self.__activated = self.mouseHovering()


    #A method which returns true if the user's mouse lies within this square.
    #A boolean mousePressed is passed through to indicate that the user has also clicked their mouse.
    def mouseHovering(self):
```

```
        mouse_x = pygame.mouse.get_pos()[0]
        mouse_y = pygame.mouse.get_pos()[1]

        if hitBoxesTouching(self.__hitbox, [(mouse_x, mouse_x), (mouse_y, mouse_y)]) and not self.__blank:
            return True
        else:
            return False

    def correctLetter(self):

        if self.__correct:
            colour = GREEN
        else:
            colour = RED

        self.__letter_object = Text("chalkboard", 25, colour, [self.center_x, self.center_y], self.__letter_answer

    def update(self):

        if not self.__blank:

            #If activated then draw a blue rectangle to indicate to the user that this square is active.
            if self.__activated:
                pygame.draw.rect(screen, BLUE, self.__rect_values, width=3)

            if self.__letter_object != "":
                self.__letter_object.update()
            if self.__number_object != "":
                self.__number_object.update()

        else:
            #A square filled with black.
            pygame.draw.rect(screen, BLACK, self.__rect_values)


#A class which represents the crossword.

class Crossword():

    def __init__(self, grid, nodes):

        #The 2D array generated by crossword_generator.py and passed through in the __init__ method.
        self.__grid = grid

        #The nodes passed through from crossword_generator.py.
        self.__nodes = nodes

        self.__grid_size = len(self.__grid)
        self.__height = SCREEN_SIZE_Y / 1.5
        self.__width = self.__height
        self.__coords = [SCREEN_SIZE_X/10, SCREEN_SIZE_Y/5.5]

        #The number of pixels between each GridSquare().
        self.__line_spaces = self.__width / self.__grid_size

        #A list of tuples in the format [((start_x, start_y), (end_x, end_y)), ...]
        #which contain the neccessary details to draw all of the lines.
        self.__line_values = []

        #A 2D array which will contain the GridSquare() objects.
        self.__grid_squares = [[] for _ in range(self.__grid_size)]

        #A boolean which indicates whether the check-answers button has been pressed.
        self.__answers_checked = False

        #This chunk of code fills the line_values list with all of the neccessary values.
        vertical_line_space = 0
        horizontal_line_space = 0
        for i in range(self.__grid_size + 1):

            self.__line_values.append(((self.__coords[0] + vertical_line_space, self.__coords[1]),
                                       (self.__coords[0] + vertical_line_space, self.__coords[1] + self.__height)))
            vertical_line_space += self.__line_spaces

            self.__line_values.append(((self.__coords[0], self.__coords[1] + horizontal_line_space),
                                       (self.__coords[0] + self.__width, self.__coords[1] + horizontal_line_space)))
            horizontal_line_space += self.__line_spaces
```

```
        #Fills the grid_squares 2D array.
        for row in range(len(self.__grid)):
            for col in range(len(self.__grid[row])):

                #Uses self.line_spaces to indicate how wide and high the rectangle should be.
                #2 is added to account for the line width.
                rect_values = [self.__coords[0] + (self.__line_spaces * col), self.__coords[1] + (self.__line_space

                #Puts the GridSquare() object into the correct row of the 2D array.
                self.__grid_squares[row].append(GridSquare(rect_values, grid[row][col]))

        #This for loop adds numbers to the GridSquare() objects where neccessary.
        for node in self.__nodes:

            if self.__grid_squares[node.getStartRow()][node.getStartCol()].getNumber() != 0:
                self.__grid_squares[node.getStartRow()][node.getStartCol()].addNumber(self.__grid_squares[node.getS
            else:
                self.__grid_squares[node.getStartRow()][node.getStartCol()].addNumber(node.getNumber())

            node.grid_square_number = self.__grid_squares[node.getStartRow()][node.getStartCol()].getNumber()

        self.__clue_list = ClueList(self.__nodes)

    def getNodes(self):
        return self.__nodes

    def getGridSquares(self):
        return self.__grid_squares

    #A method which is ran if one of the arrow keys is pressed and shifts which GridSquare() is activated.

    def arrowKey(self, row_increase, col_increase):

        #This chunk finds which GridSquare is currently activated.
        a_grid_activated = False
        for row in range(len(self.__grid_squares)):
            for col, grid_square in enumerate(self.__grid_squares[row]):
                if grid_square.getActivated():
                    a_grid_activated = True
                    grid_square_row = row
                    grid_square_col = col
                    grid_square_activated = grid_square

        #This chunk changes the activated GridSquare() to one of the adjacent ones depending on which arrow key is
        if a_grid_activated:
            new_row = grid_square_row + row_increase
            new_col = grid_square_col + col_increase
            if (new_row < len(self.__grid_squares)) and (new_col < len(self.__grid_squares[grid_square_col])):
                if not self.__grid_squares[new_row][new_col].getBlank():
                    grid_square_activated.setActivated(False)
                    self.__grid_squares[new_row][new_col].setActivated(True)


    #This method checks which word-lines and letters are correct and returns the number of correct answers.

    def checkAnswers(self):

        self.__answers_checked = True

        for node in self.__nodes:

            #Assumes the node is correct.
            node.setCorrect(True)

            if node.getDirection() == "down":

                for row in range(node.getStartRow(), node.getEndRow()+1):
                    #Compares the letter in the grid square to the corresponing letter in the expected answer provi
                    if self.__grid_squares[row][node.getStartCol()].getLetterEntered() != node.getFillingInWord()[
                        node.setCorrect(False)
                        self.__grid_squares[row][node.getStartCol()].setCorrect(False)
                        self.__grid_squares[row][node.getStartCol()].correctLetter()
            else:

                for col in range(node.getStartCol(), node.getEndCol()+1):
                    if self.__grid_squares[node.getStartRow()][col].getLetterEntered() != node.getFillingInWord()[
```

```
                        node.setCorrect(False)
                        self.__grid_squares[node.getStartRow()][col].setCorrect(False)
                        self.__grid_squares[node.getStartRow()][col].correctLetter()

        num_correct = 0
        for node in self.__nodes:
            if node.getCorrect():
                num_correct += 1

        return num_correct

    def mouseHovering(self):

        hovering = False

        for row in range(len(self.__grid_squares)):
            for square in self.__grid_squares[row]:

                if square.mouseHovering():
                    hovering = True

        return hovering

    def mouseClick(self):

        for row in range(len(self.__grid_squares)):
            for square in self.__grid_squares[row]:

                square.mouseClick()

    def update(self):

        #Draws all of the lines.
        for coord_set in self.__line_values:
            pygame.draw.line(screen, BLACK, coord_set[0], coord_set[1], width = 2)

        #updates_last is used because the blue activated rectangle needs to be displayed above the crossword
        update_last = ""

        for row in range(len(self.__grid_squares)):
            for grid_square in self.__grid_squares[row]:

                if grid_square.getActivated():
                    update_last = grid_square
                else:
                    grid_square.update()

        if update_last != "":
            update_last.update()

        self.__clue_list.update()


#A class which handles every bit of text used in the interface.

class Text():

    def __init__(self, font_name, font_size, colour, coords, text_string, x_centering=True, y_centering=True, bold=

        self.__text_string = text_string
        self.__colour = colour
        self.__coords = coords
        self.__font_name = font_name
        self.__font_size = font_size

        self.__displaying = True
        self.__x_centering = x_centering
        self.__y_centering = y_centering
        self.__bold = bold
        self.__underlined = underlined

        self.update()

    def setColour(self, colour):
        self.__colour = colour

    def setDisplay(self, bool):
```

```
        self.__displaying = bool

    def setTextString(self, text_string):
        self.__text_string = text_string

    def getTextRect(self):
        return self.__text_rect

    def getDisplaying(self):
        return self.__displaying

    def getCoords(self):
        return self.__coords

    def update(self):

        #Intialise the font_object.
        self.font_object = pygame.font.SysFont(self.__font_name, self.__font_size, bold=self.__bold)
        if self.__underlined:
            self.font_object.set_underline(True)

        #Render the font_object.
        self.font_object_render = self.font_object.render(self.__text_string, True, self.__colour)

        #Get the coords of the center of the text.
        self.center_coords = self.font_object_render.get_rect(center=self.__coords)

        #This chunk creates a list called text_rect to represent a box around the text
        #in the format (x_pos, y_pos, width, height). The box is 15 pixels larger than the text.
        text_rect_font_object = pygame.font.SysFont(self.__font_name, self.__font_size + 15, bold=self.__bold)
        text_rect_width, text_rect_height = text_rect_font_object.size(self.__text_string)
        text_rect_font_object = text_rect_font_object.render(self.__text_string, True, self.__colour)
        self.__text_rect = [text_rect_font_object.get_rect(center=self.__coords)[0], text_rect_font_object.get_rect

        #This chunk deals with centering.
        if not self.__x_centering:
            self.center_coords[0] = self.__coords[0]
            self.__text_rect[0] = self.__coords[0]
        if not self.__y_centering:
            self.center_coords[1] = self.__coords[1]
            self.__text_rect[1] = self.__coords[1]

        #Finally, display the text on the screen.
        if self.__displaying:
            screen.blit(self.font_object_render, self.center_coords)


#A class which represents a collection of Button() objects.

class ButtonSet():

    def __init__(self, button_titles, button_set_number, button_set_title):

        self.__button_amount = len(button_titles)
        self.__buttons = []
        self.__button_set_title = button_set_title

        #These are the values outputed once the options have been selected by the user.
        self.__output_values = []

        #This is a dictionary which converts the button titles to appropiate output values for the crossword_genera
        self.__output_values_conversion = {"Unit 1":"diversite", "Unit 2":"marginalise", "Unit 3":"criminels", "Uni
                                           "8x8":8, "10x10":10, "12x12":12, "14x14":14, "French":"French", "English":

        x_val = SCREEN_SIZE_X / 3
        y_val = (SCREEN_SIZE_Y / 5) * button_set_number

        self.__title = Text("chalkboard", 27, BLUE, (30, y_val), self.__button_set_title, False, True)

        #Creates the Button() objects for this ButtonSet().
        for i in range(self.__button_amount):
            self.__buttons.append(Button((x_val, y_val), button_titles[i]))
            x_val += SCREEN_SIZE_X/5

    def getOutputValues(self):
        return self.__output_values
```

```
    def getButtons(self):
        return self.__buttons

    def updateObjects(self):

        for i, button in enumerate(self.__buttons):

            button.update()

            #This chunk decides what the output_values are.
            if button.getPressed() and self.__output_values_conversion[button.getTitle()] not in self.__output_valu
                self.__output_values.append(self.__output_values_conversion[button.getTitle()])
            elif not button.getPressed() and self.__output_values_conversion[button.getTitle()] in self.__output_va
                self.__output_values.remove(self.__output_values_conversion[button.getTitle()])

        self.__title.update()


#A class for all the buttons.

class Button():

    def __init__(self, coords, title, font_size = 18, image=button_sprite_1, using_text=True, centering=True, non_t

        self.__coords = coords
        self.__displaying = True
        self.__title = title
        self.__font_size = font_size
        self.__text_colour = BLACK
        self.__image = image
        self.__centering = centering
        self.__pressed = False

        #This boolean indicates that the button will disappear once it is pressed.
        self.__non_toggle = non_toggle

        #This boolean indicates whether the button has the need for a Text() object for its title.
        self.__using_text = using_text

        self.__text = Text("chalkboard", self.__font_size, self.__text_colour, self.__coords, self.__title, self.__

        self.__width, self.__height = self.__text.getTextRect()[2], self.__text.getTextRect()[3]

        self.__hitbox = [(self.__text.getTextRect()[0], self.__text.getTextRect()[0] + self.__width), (self.__text.

    def mouseHovering(self, mouse_pressed):

        if self.__displaying:

            mouse_x = pygame.mouse.get_pos()[0]
            mouse_y = pygame.mouse.get_pos()[1]

            if hitBoxesTouching(self.__hitbox, [(mouse_x, mouse_x), (mouse_y, mouse_y)]):

                if mouse_pressed:
                    self.togglePressed()
                return True

            else:
                return False

    def reset(self):

        self.__displaying = True
        self.__pressed = False

    def getTitle(self):
        return self.__title

    def getPressed(self):
        return self.__pressed

    def togglePressed(self):

        if not self.__non_toggle:

            if self.__pressed:
```

```
                self.__pressed = False
                self.__text.setColour(BLACK)
            else:
                self.__pressed = True
                self.__text.setColour(GREEN)

        else:

            self.__pressed = True
            self.__displaying = False

    def update(self):

        if self.__displaying:

            screen.blit(pygame.transform.smoothscale(self.__image, (self.__width, self.__height)), (self.__text.get

            if self.__using_text:
                self.__text.update()


#A class for the menu screen.

class MenuScreen():

    def __init__(self):

        self.__program_loop = True

        #A boolean which indicates whether to display this screen or not.
        self.__current_screen = False

        self.__button_sets = [ButtonSet(["8x8","10x10","12x12","14x14"], 1, "Select Grid Dimension(s):"),
                              ButtonSet(["Unit 1","Unit 2","Unit 3"], 2, "Select Topic(s):"),
                              ButtonSet(["French","English"], 3, "Select a Language(s):")]

        #A Text() object for the title of the MenuScreen().
        self.__title = Text("chalkboard", 52, BLUE, (SCREEN_SIZE_X/2, 30), "French Crossword Game", bold=True, unde

        #A Button() object for the button you use to submit your options.
        self.__begin_button = Button([SCREEN_SIZE_X/2, SCREEN_SIZE_Y/1.1], "Create Crossword", 20, button_sprite_2,


    # A method which checks and handles all of the possible events and user inputs.

    def __checkEvents(self):

        #This chunk checks if the mouse is hovering over something clickable.
        hovering = False
        for button_set in self.__button_sets:
            for button in button_set.getButtons():
                if button.mouseHovering(False):
                    hovering = True
        if self.__begin_button.mouseHovering(False):
            hovering = True

        #Change the mouse to its "hand" form for if hovering is True.
        if hovering:
            pygame.mouse.set_cursor(pygame.SYSTEM_CURSOR_HAND)
        else:
            pygame.mouse.set_cursor(pygame.SYSTEM_CURSOR_ARROW)

        #Goes through all of the current input events from the user.
        for event in pygame.event.get():

            #If the user presses the red exit button of the window then quit the program.
            if event.type == pygame.QUIT:
                self.__current_screen = False
                self.__program_loop = False

            #If the mouse is clicked then check if it impacts a button.
            elif event.type == pygame.MOUSEBUTTONDOWN:
                for button_set in self.__button_sets:
                    for button in button_set.getButtons():
                        button.mouseHovering(True)
                self.__begin_button.mouseHovering(True)
```

```
        #If the begin_button has been pressed then change to the CrosswordScreen().
        if self.__begin_button.getPressed():
            self.__begin_button.togglePressed()
            self.__current_screen = False

    def __updateObjects(self):

        screen.fill(GREY)

        self.__title.update()

        for button_set in self.__button_sets:
            button_set.updateObjects()

        self.__begin_button.update()

        pygame.display.update()
        fps_clock.tick(FPS)


    #A method which loops while this object is the screen.

    def run(self):

        self.__current_screen = True

        while self.__current_screen:

            self.__checkEvents()

            self.__updateObjects()

        output_values = []
        for button_set in self.__button_sets:
            output_values.append(button_set.getOutputValues())

        return self.__program_loop, output_values


#A class for the crossword screen.

class CrosswordScreen():

    def __init__(self):

        self.__current_screen = False
        self.__program_loop = True

        #A Button() object for the button which returns you to the MenuScreen().
        self.__back_button = Button([0, 0], "Back Button", 20, button_sprite_3, False, False)

        #A Button() object for the non-toggle button which you press to check your answer.
        self.__check_button = Button([SCREEN_SIZE_X - 120, 20], "Check Crossword", non_toggle=True)

        #The text which displays the user's score once they have checked their answers.
        self.__score_text = Text("chalkboard", 52, BLUE, (SCREEN_SIZE_X - 100, 30), "")
        self.__score_text.displaying = False


    #A method which resets some of the CrossWordScreen() attributes once you have finished with the current crossw

    def __reset(self):

        self.__number_correct = -1
        self.__check_button.reset()
        self.__score_text.setDisplay(False)

    def __checkEvents(self):

        hovering = self.crossword.mouseHovering()

        if self.__back_button.mouseHovering(False) or self.__check_button.mouseHovering(False):
            hovering = True

        if hovering:
            pygame.mouse.set_cursor(pygame.SYSTEM_CURSOR_HAND)
        else:
```

```
            pygame.mouse.set_cursor(pygame.SYSTEM_CURSOR_ARROW)

        #If the check_button is pressed then check the answers and display the score_text.
        if self.__check_button.getPressed() and not self.__score_text.getDisplaying():
            self.__number_correct = self.crossword.checkAnswers()

            self.__score_text.setTextString(f"{self.__number_correct}/{len(self.crossword.getNodes())}")
            self.__score_text.setDisplay(True)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:
                self.__current_screen = False
                self.__program_loop = False

            elif event.type == pygame.MOUSEBUTTONDOWN:

                self.__back_button.mouseHovering(True)
                self.__check_button.mouseHovering(True)
                self.crossword.mouseClick()

            elif event.type == pygame.KEYDOWN:

                row_increase = 0
                col_increase = 0
                if event.key == pygame.K_UP:
                    row_increase = -1
                    self.crossword.arrowKey(row_increase, col_increase)
                elif event.key == pygame.K_DOWN:
                    row_increase = 1
                    self.crossword.arrowKey(row_increase, col_increase)
                elif event.key == pygame.K_LEFT:
                    col_increase = -1
                    self.crossword.arrowKey(row_increase, col_increase)
                elif event.key == pygame.K_RIGHT:
                    col_increase = 1
                    self.crossword.arrowKey(row_increase, col_increase)

                #This handles if a letter is typed into a GridSquare().
                elif event.unicode != "":

                    #Checks if it is a lower-case letter of the alphabet.
                    if ord(event.unicode) >= 97 and ord(event.unicode) <= 122:

                        for row in range(len(self.crossword.getGridSquares())):
                            for square in self.crossword.getGridSquares()[row]:

                                square.keyDown(event.unicode)

        #If the back_button is pressed then return to the MenuScreen().
        if self.__back_button.getPressed():
            self.__back_button.togglePressed()
            self.__current_screen = False

    def __updateObjects(self):

        screen.fill(WHITE)

        self.__back_button.update()
        self.__check_button.update()
        self.crossword.update()

        self.__score_text.update()

        pygame.display.update()
        fps_clock.tick(FPS)

    def run(self, input_values):

        self.__reset()

        self.input_values = input_values

        if input_values[0] == []:
            self.input_values[0] = [8, 10, 12, 14]
        crossword_size = choice(self.input_values[0])
```

```
        if input_values[1] == []:
            self.input_values[1] = ["diversite", "marginalise", "criminels"]
        topics = self.input_values[1]

        if input_values[2] == []:
            self.input_values[2] = ["French", "English"]
        language = choice(self.input_values[2])

        self.__current_screen = True
        grid, nodes = main(crossword_size, topics, language)
        self.crossword = Crossword(grid, nodes)

        while self.__current_screen:

            self.__checkEvents()

            self.__updateObjects()

        return self.__program_loop


#This function compares two hitbox lists to see if they intersect with each other and return True if they do.

def hitBoxesTouching(hitbox_1, hitbox_2):

    x_touching = True
    y_touching = True

    if hitbox_1[0][0] > hitbox_2[0][1] or hitbox_1[0][1] < hitbox_2[0][0]:
        x_touching = False

    if hitbox_1[1][1] < hitbox_2[1][0] or hitbox_1[1][0] > hitbox_2[1][1]:
        y_touching = False

    if y_touching and x_touching:
        return True
    else:
        return False


 #This procedure keeps looping and switching between the menu and crossword screens untill the program is quitted.

def programLoop(program_loop):

    menu_screen = MenuScreen()
    crossword_screen = CrosswordScreen()

    while program_loop:

        program_loop, output_values = menu_screen.run()

        if program_loop:
            program_loop = crossword_screen.run(output_values)

if __name__ == "__main__":

    programLoop(True)

pygame.quit()
```