Multimedia Systems

Project Report

Romy S Peter
Riya Rajesh
Emma Mary Cyriac
Hemanth Sai Borra

MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE

TABLE OF CONTENTS

Page #

1.0 GENERAL INFORMATION

1.1 Problem Statement

Games have always been an integral part of a person's early childhood development. It is a source of entertainment, encourages creativity and provides a means to take a break from the pressures of life.

The "shoot-em-up" genre of games, which includes the game 'Space Invaders' is a time untold classic. It was one of the forerunners of modern video gaming and helped expand the video game industry from a novelty to a global industry.

However, given its age and analogue arcade status these days, we wanted to give it a digital makeover for the new age.

1.2 Abstract

Space Invaders was developed by Tomohiro Nishikado and released in 1978. It was one of the earliest shooting games and the aim is to defeat waves of aliens with a laser cannon to earn as many points as possible.

When first released, Space Invaders was very successful. The game has been the inspiration for other video games, re-released on numerous platforms, and led to several sequels.

Ever since, it has become sort of a pop culture icon, often used as a synecdoche representing video games as a whole.

1.3 Introduction to the Project

The general objective of this project was to develop a 2D game that has replay value, a good foundation about game mechanics and gameplay immersion while also employing something we've learned in our Multimedia Systems course. As such, we've chosen to make a game which includes all forms of media like images, text and audio.

Some core functionalities implemented that correspond to what we've learnt:

- Using custom fonts
- Image manipulation
- Dealing with audio files (particularly .WAV)
- Image format selection
- Rendering various forms of media
- Animation

2.0 SYSTEM REQUIREMENTS

2.1 Hardware & Software Requirements

For the successful, efficient and error-free functioning of any program that involves graphics, the system should meet some requirements. In this section, the various requirements that are necessary for this program are specified.

- Hardware requirements

    - CPU: Any x64-based processor
    - RAM: 1 GB or more
    - Graphics memory: 64 MB or more
    - Storage: 40 GB or more

- Software requirements

    - OS: Windows
    - Platform / Language: Python (with Pygame)
    - Software(s): Python 3.10 (64-bit), Pygame, Source code editor (if required).

2.2 Installation Instructions

The program is pretty straightforward to run. It requires Python 3.10 (necessary) and Pygame to run properly. Python can be downloaded and installed on Windows from the official website. Following this, Pygame can be installed using the command:

```
python -m pip install -U pygame --user
```

- To compile the program, cd to the directory it is stored in.

- Subsequently, to run the program, type "`python main.py`"

- The rules and further instructions are mentioned in the game for reference.

3.0 SOURCE CODE

3.1 Directory Structure

We have made sure to document our source code very thoroughly and ensured it is of very high quality. Before moving on to the code, the directory structure will be briefly discussed below.

```
Space Impact/
├── assets/
│   ├── fonts/
│   ├── graphics/
│   └── sounds/
├── models/
│   ├── button.py
│   ├── controls.py
│   ├── explosion.py
│   ├── icon_button.py
│   ├── laser.py
│   ├── scores.py
│   ├── ship.py
│   └── slider.py
├── screens/
│   ├── background.py
│   ├── controls.py
│   ├── game.py
│   ├── score_board.py
│   ├── settings.py
│   └── ships.py
├── utils/
│   ├── assets.py
│   ├── collide.py
│   ├── outlineImage.py
│   └── resource_path.py
├── config.py
├── constants.py
├── LICENSE.txt
└── main.py
```

To the left is the directory structure for our project.

a) **Assets:** The assets folder contains the fonts; graphics files and sounds being used in this project. Since it contains a lot of files, only folder layout is shown

b) **Models:** The models folder contains code for various interactive components being featured in the game – which include buttons, explosions, the laser weapon system and so on. Also contains logic for stuff like the scoreboard, spaceship related functions, volume control slider.

c) **Screens:** Every module in our game is actually represented on different screens, which are referenced in the main function. Essentially, the screens folder contains the code logic for displaying these screens which include the actual gameplay screen, the control information screen, spaceship information screen, scoreboard screen, settings screen and so on. Apart from these, it also contains the logic for the scrolling background image.

d) **Utils:** Our program frequently requires the use of various common functions. The code for these functions are defined in the utils folder. This includes the outline creator, resource path setter, collision detector and so on.

e) Finally, we have 3 .py files in the root folder: main.py, which is the main program, constants.py which contains path to assets being used and config.py, which has default configuration settings used by the game.

3.2 Code

```
# Filename: ./main.py

# Function: Main program file that is used to launch the game.

# Importing all modules required for proper functioning of the code in this file.

import sys
import pygame
import argparse

# Importing the required code from other modules of the game.

from utils.assets import Assets
from screens.game import game
from screens.controls import controls
from screens.score_board import score_board
from screens.ships import ships
from screens.settings import settings
from screens.background import slow_bg_obj
from models.button import Button
from models.icon_button import IconButton
from models.controls import audio_cfg, display_cfg
from config import config
from constants import Path, Image, Font, Colors, Text

# Parses command line arguments, if any. In this case, it is disabling all audio.

ag = argparse.ArgumentParser()
ag.add_argument('--mute', help='disable all sounds', action='store_true' )
args = vars(ag.parse_args())

# Toggle mute (disable all audio) if the mute argument is passed via command
line.

if args['mute']:
    audio_cfg.toggle_mute()

# Used to initializes the font module. This must be initialized before any font
related functions can be used.

pygame.font.init()
```

```
# If the display has a window title, this function will change the name on the
window.

pygame.display.set_caption(config.TITLE)

# Sets the runtime icon the system will use to represent the display window. In
this case, we use a 32x32 crop of the player's ship.

pygame.display.set_icon(Image.PLAYER_SPACE_SHIP)


# Execution of program begins here.

def main():

    # Sets various parameters for the two fonts being loaded (such as the font
size).

    title_font = pygame.font.Font(Font.edit_undo_font, 60)  # Title font (used to
show 'Start Game')
    grp_font = pygame.font.Font(Font.edit_undo_font, 30)  # Group font (used to
show 'Designed by RERH - MS Group 4').

    # Plays the menu music.

    audio_cfg.play_music(Path.MENU_MUSIC_PATH)

    # Loads the button images from the assets folder for buttons using images or
button color for primitive buttons.
    # In the below list, the 'MOUSE' and 'KEYBOARD' buttons are just white
bordered rectangles with text in them.
    # Rest are buttons that loads as images from assets folder.

    mouse_btn = Button(Colors.BACKGROUND_BLACK, Colors.WHITE, 'MOUSE')
    keyboard_btn = Button(Colors.BACKGROUND_BLACK, Colors.WHITE, 'KEYBOARD')
    control_btn = IconButton(Image.CONTROL_IMAGE, Text.CONTROLS)
    ships_btn = IconButton(Image.SHIPS_IMAGE, Text.SHIPS)
    trophy_btn = IconButton(Image.TROPHY_IMAGE, Text.SCOREBOARD)
    settings_btn = IconButton(Image.TOOLBOX_IMAGE, Text.SETTINGS)
    exit_btn = IconButton(Image.EXIT_IMAGE)
```

```
    # Loop that runs the current screen. Value of run changes to false when game
begins, another screen is called or if user quits the game.

    run = True
    while run:

        # Shows the mouse cursor

        pygame.mouse.set_visible(True)

        # Updates the background image (i.e. space) and renders it dynamically
(i.e. screen size changes for example).

        slow_bg_obj.update()
        slow_bg_obj.render()

        # Draws the text 'Designed by RERH - MS Group 4' at (y+250) from the
center (x).

        Assets.text.draw(
            'Designed by RERH - MS Group 4',
            grp_font,
            Colors.WHITE,
            (config.center_x, config.center_y + 250),
            True,
            True,
            )

        # Draws the mouse and keyboard buttons at (x-210, y+42) from center of
screen.

        mouse_btn.draw((config.center_x - 210, config.center_y + 42), (195, 66))
        keyboard_btn.draw((config.center_x + 15, config.center_y + 42), (195,
66))

        # Draws the text 'Start Game' at (y-10) from the center (x).

        Assets.text.draw(
            'Start Game',
            title_font,
            Colors.WHITE,
            (config.center_x, config.center_y - 10),
            True,
            True,
            )
```

```
# Draws the control page button at (x+10, 53).
        control_btn.draw((config.starting_x + 65, 53), True, True)
        # Draws the scoreboard page button at (x-65, 55).
        trophy_btn.draw((config.ending_x - 65, 55), True, True)
        # Draws the settings page button at (x-65, 165).
        settings_btn.draw((config.ending_x - 65, 165), True, True)
        # Draws the ships page button at (x+65, 165).
        ships_btn.draw((config.starting_x + 65, 165), True, True)
        # Displays the volume at the bottom left of the screen.
        audio_cfg.display_volume()
        # Draws the exit button at (x-65, 165).
        exit_btn.draw((config.ending_x - 75, config.ending_y - 40), True, True)
        # Draws the logo of the game on the top at y=50 from center (x).
        Assets.image.draw(Image.TITLE_LOGO, (config.center_x, 50), True)
        # Updates the content on the entire screen.
        pygame.display.flip()
        # Caps the framerate to 60 for a smooth experience.
        config.clock.tick(config.FPS)
        # Used to quit the game in case user clicks on the 'X' button on the
window.
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
```

```
# Used to dynamically scale the window if the user uses the maximize button on
the window.

            if event.type == pygame.VIDEORESIZE:
                if not display_cfg.fullscreen:
                    config.update(event.w, event.h)

            # Responds to keyboard input events, such as reducing / increasing
volume, disabing audio, or toggling full screen.

            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_m:
                    audio_cfg.toggle_mute()
                if event.key == pygame.K_PLUS or event.key == pygame.K_EQUALS:
                    audio_cfg.inc_volume(5)
                if event.key == pygame.K_MINUS:
                    audio_cfg.dec_volume(5)
                if event.key == pygame.K_f:
                    config.update(config.monitor_size[0],
                                  config.monitor_size[1])
                    display_cfg.toggle_full_screen()

            # Responds to mouse click events and redirects to different screens
depending on the button clicked.

            if event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                    if mouse_btn.isOver():
                        game(True)
                    if keyboard_btn.isOver():
                        game()
                    if control_btn.isOver():
                        controls()
                    if trophy_btn.isOver():
                        score_board()
                    if ships_btn.isOver():
                        ships()
                    if settings_btn.isOver():
                        settings()
                    if exit_btn.isOver():
                        run = False
```

```
            # Responds to mouse hover events and shows a white outline on buttons
or any interactive elements the mouse hovers over.

            if event.type == pygame.MOUSEMOTION:
                if mouse_btn.isOver():
                    mouse_btn.outline = True
                else:
                    mouse_btn.outline = False

                if keyboard_btn.isOver():
                    keyboard_btn.outline = True
                else:
                    keyboard_btn.outline = False

                if control_btn.isOver():
                    control_btn.outline = True
                else:
                    control_btn.outline = False

                if trophy_btn.isOver():
                    trophy_btn.outline = True
                else:
                    trophy_btn.outline = False

                if settings_btn.isOver():
                    settings_btn.outline = True
                else:
                    settings_btn.outline = False

                if ships_btn.isOver():
                    ships_btn.outline = True
                else:
                    ships_btn.outline = False

                if exit_btn.isOver():
                    exit_btn.outline = True
                else:
                    exit_btn.outline = False
```

```
# Breaks out of the loop and exits the game if 'Q' or 'ESC' key is pressed.

        keys = pygame.key.get_pressed()
        if keys[pygame.K_ESCAPE] or keys[pygame.K_q]:
            run = False

        # Takes user to the control information screen when 'C' key is pressed.

        if keys[pygame.K_c]:
            controls()

        # Takes user to the scoreboard screen when 'S' key is pressed.

        if keys[pygame.K_s]:
            score_board()

    # Deactivates the pygame library and then terminates the program with code 0,
which indicates successful execution.

    pygame.quit()
    sys.exit(0)


main()
```

```
# Filename: ./constants.py

# Function: Constants file that contains paths to various assets that are used
in-game (such as buttons, spaceship and so on).

# Importing all modules required for proper functioning of the code in this file.

import pygame
import os

# Importing the required code from other modules of the game.

from utils.resource_path import resource_path
from utils.assets import Assets

# Array that holds the list of all sounds used in the game.

soundList = []

# Initializes the in-built sound module provided by pygame.

pygame.mixer.init()

# Specifies the folders where various assets are present.
# Fonts are present in /assets/fonts/
# Images (like arrows, buttons, spaceships, icons, logos) are present in /as-
sets/graphics/
# Explosion sequence PNG files are present in /assets/graphics/explosion
# Audio files are present in /assets/sounds/

class Path:
    FONT_PATH = os.path.join('assets', 'fonts')
    EXPLOSION_PATH = os.path.join('assets', 'graphics', 'explosion')
    GRAPHICS_PATH = os.path.join('assets', 'graphics')
    SOUND_PATH = os.path.join('assets', 'sounds')

    # Loads the menu and in-game music files.

    GAME_MUSIC_PATH = resource_path(os.path.join(SOUND_PATH, 'ingame.wav'))
    MENU_MUSIC_PATH = resource_path(os.path.join(SOUND_PATH, 'menu.wav'))
```

```
# Loads various font files that are used in the game.
# In our project, we use 3 different fonts:
# (a) Edit Undo BRK
# (b) Karmatic Arcade
# (c) Neue Pixel Sans
# Thanks & credits to all the authors of these fonts for their wonderful
creations!

class Font:
    edit_undo_font = Assets.font.load(Path.FONT_PATH, 'edit_undo.ttf')
    neue_font = Assets.font.load(Path.FONT_PATH, 'neue.ttf')
    karmatic_arcade_font = Assets.font.load(Path.FONT_PATH,
'karmatic_arcade.ttf')

# Loads all the graphics files that are used in the game. First one loads the
logo of the game, shown on the menu.
class Image:
    TITLE_LOGO = Assets.image.scale(Path.GRAPHICS_PATH, 'title_logo.png', 2/7)

    # Loads the graphics assets for all enemy spaceships used in the game.

    EASY_SPACE_SHIP = Assets.image.load(Path.GRAPHICS_PATH, 'easy.png')
    MEDIUM_SPACE_SHIP = Assets.image.load(Path.GRAPHICS_PATH, 'medium.png')
    HARD_SPACE_SHIP = Assets.image.load(Path.GRAPHICS_PATH, 'hard.png')
    BOSS_SHIP = Assets.image.load(Path.GRAPHICS_PATH, 'boss.png')

    UFO_SPACE_SHIP = Assets.image.scale(Path.GRAPHICS_PATH, 'ufo.png', 1/7)

    # Loads the graphics assets for the user's spaceship and it's laser weapon.

    PLAYER_SPACE_SHIP = Assets.image.load(
        Path.GRAPHICS_PATH, 'retro-spaceship.png')
    PLAYER_LASER = Assets.image.load(
        Path.GRAPHICS_PATH, 'pixel_laser_cosmic.png')

    # Loads the graphics assets for the enemy spaceship's laser weapons.

    RED_LASER = Assets.image.load(Path.GRAPHICS_PATH, 'pixel_laser_red.png')
    BLUE_LASER = Assets.image.load(Path.GRAPHICS_PATH, 'pixel_laser_blue.png')
    GREEN_LASER = Assets.image.load(
        Path.GRAPHICS_PATH, 'pixel_laser_green.png')
    FLAME_LASER = Assets.image.load(
        Path.GRAPHICS_PATH, 'pixel_laser_flame.png')
```

```
# Load the graphic assets for audio related icons that are used in the game.

    VOL_ICON = Assets.image.load(Path.GRAPHICS_PATH, 'audio.png')
    MUTE_ICON = Assets.image.load(Path.GRAPHICS_PATH, 'mute.png')

    DEMON_ICON = Assets.image.scale(Path.GRAPHICS_PATH, 'demon.png', 1/11)

    # Loads the graphic assets for icons / buttons that appear in the menu (such
as settings, scoreboard, controls).

    CONTROL_IMAGE = Assets.image.load(Path.GRAPHICS_PATH, 'joystick.png')
    TROPHY_IMAGE = Assets.image.load(Path.GRAPHICS_PATH, 'trophy.png')
    SHIPS_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'medium.png', 5/6)
    SHIPS_IMAGE_2 = Assets.image.scale(Path.GRAPHICS_PATH, 'hard.png', 3/4)
    TOOLBOX_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'toolbox.png', 1/2)

    TOOLS_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'tools.png', 1/4)
    TOOLS_IMAGE = pygame.transform.rotate(TOOLS_IMAGE, -45)

    GO_BACK_IMAGE = Assets.image.scale(
        Path.GRAPHICS_PATH, 'back_arrow.png', 6/25)

    EXIT_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'exit_button.png', 1/3)

    # Loads the graphic assets for various other icons that are used throughout
gameplay (such as scores, pause, lives).

    HEART_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'heart.png', 1)
    STAR_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'star.png', 1/4)
    SKULL_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'skull.png', 1/58)
    SKULL_IMAGE_2 = Assets.image.scale(Path.GRAPHICS_PATH, 'skull.png', 1/54)
    WON_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'won.png', 5/20)

    PLUS_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'plus.png', 1/6)
    MINUS_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'minus.png', 1/6)

    PAUSE_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'pause.png', 2/7)
    PLAY_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'play.png', 2/7)
    PLAY_IMAGE_2 = Assets.image.scale(Path.GRAPHICS_PATH, 'play.png', 1/2.9)
```

```
HOME_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'home.png', 2/5)
    NEXT_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'next_button.png', 1/3)
    BACK_IMAGE = Assets.image.scale(Path.GRAPHICS_PATH, 'back_button.png', 1/3)
    LEVELS_IMAGE = Assets.image.scale(
        Path.GRAPHICS_PATH, 'levels_button.png', 1/3)
    SCORE_IMAGE = Assets.image.scale(
        Path.GRAPHICS_PATH, 'score_button.png', 1/3)
    KILLS_IMAGE = Assets.image.scale(
        Path.GRAPHICS_PATH, 'kills_button.png', 1/3)

    MOUSE = Assets.image.scale(Path.GRAPHICS_PATH, 'mouse.png', 1/2)
    LEFT_MOUSE_CLICK = Assets.image.scale(
        Path.GRAPHICS_PATH, 'left_click_mouse.png', 1/2)
    RIGHT_MOUSE_CLICK = Assets.image.scale(
        Path.GRAPHICS_PATH, 'right_click_mouse.png', 1/2)

    WASD_KEYS = Assets.image.scale(Path.GRAPHICS_PATH, 'wasd_keys.png', 1/2)
    ARROW_KEYS = Assets.image.scale(Path.GRAPHICS_PATH, 'arrow_keys.png', 1/2)
    BACKSPACE_KEY = Assets.image.scale(
        Path.GRAPHICS_PATH, 'backspace_key.png', 1/2)
    SPACEBAR_KEY = Assets.image.scale(
        Path.GRAPHICS_PATH, 'spacebar_key.png', 1/2)
    PLUS_KEY = Assets.image.scale(Path.GRAPHICS_PATH, 'plus_key.png', 1/2)
    MINUS_KEY = Assets.image.scale(Path.GRAPHICS_PATH, 'minus_key.png', 1/2)
    P_KEY = Assets.image.scale(Path.GRAPHICS_PATH, 'p_key.png', 1/2)
    F_KEY = Assets.image.scale(Path.GRAPHICS_PATH, 'f_key.png', 1/2)
    M_KEY = Assets.image.scale(Path.GRAPHICS_PATH, 'mute_key.png', 1/2)

# Loads the audio files associated with various actions (such as the player /
enemy firing their laser weapons, explosions and so on).

class Sound:
    PLAYER_LASER_SOUND = Assets.sound.load(Path.SOUND_PATH, 'ownlaser.wav')
    ENEMY_LASER_SOUND = Assets.sound.load(Path.SOUND_PATH, 'enemylaser.wav')
    EXPLODE_SOUND = Assets.sound.load(Path.SOUND_PATH, 'explode.wav')
    LASER_HIT_SOUND = Assets.sound.load(Path.SOUND_PATH, 'laser_hit.wav')


# Adds the list of sounds to the array we declared earlier.

soundList.append(Sound.PLAYER_LASER_SOUND)
soundList.append(Sound.ENEMY_LASER_SOUND)
soundList.append(Sound.EXPLODE_SOUND)
soundList.append(Sound.LASER_HIT_SOUND)
```

```
# Defines the RGB color codes for various colors we use througout the game,
especially for fonts.

class Colors:
    WHITE = (255, 255, 255)
    BLACK = (0, 0, 0)
    BACKGROUND_BLACK = (7, 8, 16)
    BLUE = (0, 0, 255)
    GREEN = (0, 255, 0)
    GREEN2 = (0, 209, 0)
    RED = (255, 0, 0)
    YELLOW = (255, 255, 0)
    CYAN = (0, 255, 255)
    MAGENTA = (255, 0, 255)
    PURPLE = (131, 1, 123)
    ORANGE = (238, 98, 17)
    GREY = (200, 200, 200)
    TRANS = (1, 1, 1)

# Stores text in variables so as to reference them easier.

class Text:
    SHIPS = 'SHIPS'
    SETTINGS = 'SETTINGS'
    CONTROLS = 'CONTROLS'
    SCOREBOARD = 'SCOREBOARD'
```

```
# Filename: ./config.py

# Function: Configuration file that contains parameters for various settings that
the game uses.

# Importing all modules required for proper functioning of the code in this file.

import pygame
import os
import ctypes

# Importing the required code from other modules of the game.

from utils.resource_path import resource_path

# Configuration parameters are defined below.

# First one sets the title of the game window and a default resolution in-case
automatic resolution detection by pygame fails.

class Config:
    def __init__(self):

        self.TITLE = 'Space Impact'
        self.WIDTH = 750
        self.HEIGHT = 750

        # Loads the background image (i.e. image of space) from /assets/graphics.

        self.backgroundImage = pygame.image.load(resource_path(
            os.path.join('assets', 'graphics', 'background-black-wide.png')))

        # Function that returns the width and height of the screen pixels as a
two-integer tuple.

        windows_user = ctypes.windll.user32
        self.monitor_size = (windows_user.GetSystemMetrics(0),
                             windows_user.GetSystemMetrics(1))

        # Sets the dimension for the background image based on the resolution
retrieved by previous function.

        self.BG = pygame.transform.scale(
            self.backgroundImage, self.monitor_size)
```

```
# Sets the default framerate to 60 FPS for a smooth gameplay experience.

        self.FPS = 60
        self.clock = pygame.time.Clock()

        # Initializes a window / screen that is resizable with 750 x 750
resolution by default.

        self.CANVAS = pygame.display.set_mode(
            (self.WIDTH, self.HEIGHT), pygame.RESIZABLE)

        # Gets the width & height of the window in case it is resized. Achieved
by calculating the rectangular area of the surface.

        self.screen_rect = self.CANVAS.get_rect()
        self.center_x = self.screen_rect.centerx
        self.starting_x = 0
        self.ending_x = self.WIDTH

        self.center_y = self.screen_rect.centery
        self.starting_y = 0
        self.ending_y = self.HEIGHT


    # Updates the width and height of the window such that it can scale
dynamically (windowed -> full screen, for example).
    # Also responsible for resizing the background image used from 16:9 to 1:1 in
case the window is resized.
    # This is a fast scale operation that does not sample the results.

    def update(self, width, height):
        self.CANVAS = pygame.display.set_mode(
            (width, height), pygame.RESIZABLE)
        self.WIDTH = width
        self.HEIGHT = height
        self.center_x = width//2
        self.center_y = height//2
        self.ending_x = width
        self.ending_y = height
        self.BG = pygame.transform.scale(
            self.backgroundImage, (self.WIDTH, self.HEIGHT))


config = Config()
```

```
# Filename: ./screens/background.py

# Function: This file contains code to implement the scrolling background image
(i.e. space).

# It can be noted that in almost every screen, there is an image of space that is
scrolling, which also dynamically adjusts to scaling.
# This background is implemented by using a pixel art of space, which is stored
as a PNG file in the /assets/graphics folder.
# It is to be noted that the images are not directly used here. They are
referenced via the function.
# This file only contains the logic to implement the scrolling and also the
dynamic scaling adjustment.

# There are two versions of the scrolling background: one that scrolls at normal
speed and another that scrolls 1.5x slower.
# The former is used in the game screen while the latter is used in the menu &
other screens.

# Importing the required code from other modules of the game.

from config import config
from utils.assets import Assets

# In the below portion of code, the referenced image is stored in two different
variables.
# Then, for one, the resolution (bounding window) is calculated using get_rect().
The moving speed is then set.
# The update function then checks if the image has finished scrolling once by
checking it's height.
# If it did, it loops the image from the beginning. It is rendered on-screen by
the rendering function.

class ScrollBackground():
    def __init__(self, bg_img, moving_speed=3):
        self.bgimage = bg_img
        self.rectBGimg = self.bgimage.get_rect()

        self.bgY1 = 0

        self.bgY2 = - self.rectBGimg.height

        self.moving_speed = moving_speed

    def update(self):
```

```
        self.bgY1 += self.moving_speed
        self.bgY2 += self.moving_speed
        if self.bgY1 >= self.rectBGimg.height:
            self.bgY1 = - self.rectBGimg.height
        if self.bgY2 >= self.rectBGimg.height:
            self.bgY2 = - self.rectBGimg.height

    def render(self):
        Assets.image.draw(self.bgimage, (config.center_x, self.bgY1), True)
        Assets.image.draw(self.bgimage, (config.center_x, self.bgY2), True)

# Calls the function with input image and scrolling speed.

bg_obj = ScrollBackground(config.BG)
slow_bg_obj = ScrollBackground(config.BG, 1.5)
```

```
# Filename: ./screens/controls.py

# Function: This file contains the code required to display the 'Controls' page
in the game.

# The controls page essentially gives a low-down to new players on how the
controls work in the game.
# There are four sets of control that are shown per list. The 'back' and 'next'
arrows can be used to navigate b/w them.

# Importing all modules required for proper functioning of the code in this file.

import pygame
import sys

# Importing the required code from other modules of the game.

from .background import slow_bg_obj
from utils.assets import Assets
from models.icon_button import IconButton
from models.controls import audio_cfg, display_cfg
from config import config
from constants import Image, Font, Colors, Text


# Control screen definition begins below.

def controls():

    # Loop that runs the current screen. Value of run changes to false when game
begins, another screen is called or if user quits the game.

    run = True

    # As mentioned above, there are 4 sub-sections in this screen which
illustrates the controls.
    current_page = 1
    total_pages = 4

    # Loads all the fonts necessary for displaying text on the 'control' screen.

    control_title_font = pygame.font.Font(Font.edit_undo_font, 50)
    control_title_font_2 = pygame.font.Font(Font.edit_undo_font, 45)
    control_title_font_3 = pygame.font.Font(Font.edit_undo_font, 40)
    control_font = pygame.font.Font(Font.neue_font, 40)
```

```
# Loads the 'BACK' button and 'NEXT' button images along with the ARROW to go
back to the main menu.

    go_back_btn = IconButton(Image.GO_BACK_IMAGE)
    back_btn = IconButton(Image.BACK_IMAGE)
    next_btn = IconButton(Image.NEXT_IMAGE)

    # Subscreen 1 - Moving / Aiming
    # Mouse: The ship can be moved in all degrees of freedom supported using a
mouse. This is the most efficient way to control.
    # Keyboard: The W, A, S, D or UP, DOWN, LEFT, RIGHT arrows can be used to
control the ship. This is a bit tedious.
    def moveControlPage():
        Assets.text.draw('MOVE', control_title_font, Colors.GREEN,
                         (config.center_x-265, 190))
        Assets.text.draw('/ AIM:', control_title_font, Colors.GREEN,
                         (config.center_x-265, 240))

        Assets.text.draw('MOUSE', control_title_font_2, Colors.MAGENTA,
                         (config.center_x+202, 190), True)
        Assets.image.draw(Image.MOUSE, (config.center_x+202, 240), True)

        Assets.text.draw('KEYBOARD', control_title_font_2, Colors.MAGENTA,
                         (config.center_x-265, 380))
        Assets.text.draw('OR', control_font, Colors.WHITE,
                         (config.center_x-20, 445), True)
        Assets.image.draw(Image.WASD_KEYS, (config.center_x-280, 440))
        Assets.image.draw(Image.ARROW_KEYS, (config.center_x+30, 440))
```

```
# Subscreen 2 - Shooting
# Mouse: Left click fires the onboard laser weapon.
# Keyboard: Spacebar fires the onboard laser weapon.

def shootControlPage():
    Assets.text.draw('Shoot:', control_title_font, Colors.YELLOW,
                     (config.center_x-265, 190))

    Assets.text.draw('MOUSE', control_title_font_2, Colors.CYAN,
                     (config.center_x+202, 190), True)
    Assets.image.draw(Image.LEFT_MOUSE_CLICK,
                      (config.center_x+202, 240), True)

    Assets.text.draw('KEYBOARD', control_title_font_2, Colors.CYAN,
                     (config.center_x-265, 380))
    Assets.image.draw(Image.SPACEBAR_KEY, (config.center_x-20, 470))
    Assets.text.draw('SPACEBAR', control_font, Colors.WHITE,
                     (config.center_x-265, 460))


# Category 3 - Returning back home
# Mouse: Right click can be used to return to the main menu.
# Keyboard: Backspace can be used to return to the main menu.

def returnControlPage():
    Assets.text.draw('RETURN BACK', control_title_font, Colors.MAGENTA,
                     (config.center_x-265, 190))
    Assets.text.draw('TO HOME:', control_title_font, Colors.MAGENTA,
                     (config.center_x-265, 240))

    Assets.text.draw('MOUSE', control_title_font_2, Colors.ORANGE,
                     (config.center_x+202, 190), True)
    Assets.image.draw(Image.RIGHT_MOUSE_CLICK,
                      (config.center_x+202, 240), True)

    Assets.text.draw('KEYBOARD', control_title_font_2, Colors.ORANGE,
                     (config.center_x-265, 380))
    Assets.image.draw(Image.BACKSPACE_KEY, (config.center_x-20, 470))
    Assets.text.draw('BACKSPACE', control_font, Colors.WHITE,
                     (config.center_x-265, 460))
```

```
# Subscreen 4 - Miscellaneous Controls
    # + / - keys - increase / decrease audio.
    # M key - Disable audio (mute).
    # P key - Pause game.
    # F - Toggle fullscreen.

    def otherControlsPage():
        Assets.text.draw('VOLUME UP/DOWN:', control_title_font_3, Colors.GREEN,
                         (config.center_x-265, 200))
        Assets.image.draw(Image.PLUS_KEY,
                          (config.center_x+122, 190), True)
        Assets.image.draw(Image.MINUS_KEY,
                          (config.center_x+202, 190), True)

        Assets.text.draw('MUTE', control_title_font_3, Colors.GREEN,
                         (config.center_x-265, 290))
        Assets.image.draw(Image.M_KEY,
                          (config.center_x+162, 280), True)
        Assets.text.draw('PAUSE GAME', control_title_font_3, Colors.GREEN,
                         (config.center_x-265, 375))
        Assets.image.draw(Image.P_KEY,
                          (config.center_x+162, 365), True)
        Assets.text.draw('TOGGLE FULLSCREEN', control_title_font_3, Colors.GREEN,
                         (config.center_x-265, 460))
        Assets.image.draw(Image.F_KEY,
                          (config.center_x+162, 450), True)

    # Used to display the slow-scrolling background image

    while run:
        slow_bg_obj.update()
        slow_bg_obj.render()

        match current_page:
            case 1:
                moveControlPage()
            case 2:
                shootControlPage()
            case 3:
                returnControlPage()
            case 4:
                otherControlsPage()
```

```
# Draws the back and next buttons on the screen.

        Assets.text.draw(Text.CONTROLS, control_title_font, Colors.ORANGE,
                         (config.center_x, 100), True, False, True)
        Assets.image.draw(Image.CONTROL_IMAGE, (config.center_x + 125, 90))

        go_back_btn.draw((config.starting_x + 65, 50), True, True)
        back_btn.draw(
            (config.center_x-Image.BACK_IMAGE.get_width()-30, config.ending_y-
125))
        next_btn.draw((config.center_x + 30, config.ending_y - 125))



        # The following code given below are present on every screen component to
ensure smooth functioning of the game.

        # They include:

        # (a) Code to display the volume information
        # (b) Code to set the framerate to 60 FPS.
        # (c) Code to quit the game in case the user presses the 'X' button on
window.
        # (d) Code to update the window size and dimensions of the background
image in the event user resizes the window.
        # (e) Code to implement various keyboard button functions like modifying
volume, quit game, mute, toggling full screen.
        # (f) Code to respond to mouse click events on buttons, set 'run' to
false and return to main menu and switch between screens if the NEXT or BACK
buttons are pressed.
        # (g) Code to respond to mouse hover events, set the white outline over
interactive objects.

        # Each code block will be highlighted with their corresponding alphabets.

        # (a)
        audio_cfg.display_volume()

        # (b)
        pygame.display.flip()
        config.clock.tick(config.FPS)
```

```
        # (c)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit(0)

            # (d)
            if event.type == pygame.VIDEORESIZE:
                if not display_cfg.fullscreen:
                    config.update(event.w, event.h)

            # (e)
            if event.type == pygame.KEYUP:
                if event.key == pygame.K_m:
                    audio_cfg.toggle_mute()
                if event.key == pygame.K_PLUS or event.key == pygame.K_EQUALS:
                    audio_cfg.inc_volume(5)
                if event.key == pygame.K_MINUS:
                    audio_cfg.dec_volume(5)
                if event.key == pygame.K_f:
                    config.update(
                        config.monitor_size[0], config.monitor_size[1])
                    display_cfg.toggle_full_screen()
                if event.key == pygame.K_BACKSPACE:
                    run = False

            # (f)
            if event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                    if go_back_btn.isOver():
                        run = False
                    if back_btn.isOver():
                        if current_page == 1:
                            current_page = 1
                        else:
                            current_page -= 1
                    if next_btn.isOver():
                        if current_page == total_pages:
                            current_page = total_pages
                        else:
                            current_page += 1
```

```
# (g)
if event.type == pygame.MOUSEMOTION:
    if go_back_btn.isOver():
        go_back_btn.outline = True
    else:
        go_back_btn.outline = False

    if back_btn.isOver():
        back_btn.outline = True
    else:
        back_btn.outline = False

    if next_btn.isOver():
        next_btn.outline = True
    else:
        next_btn.outline = False
```

```
# Filename: ./screens/game.py

# Function: This file contains the code required to display the 'game' screen and
implement all gameplay logic.

# Since this file is responsible for the gameplay experience, it is also arguably
quite important.
# Here, we define various gameplay logic and mechanics that will be used in the
game.

# Importing all modules required for proper functioning of the code in this file.

import pygame
import sys
import time
import random

# Importing the required code from other modules of the game.

from models.ship import Player, Enemy
from models.explosion import Explosion, explosion_group
from models.controls import audio_cfg, display_cfg
from models.scores import scores
from models.icon_button import IconButton
from utils.collide import collide
from utils.assets import Assets
from .background import bg_obj
from config import config
from constants import Path, Image, Font, Colors

# Maintains the state for whether the game is paused or not. Since the game has
only begun, pause is set to false initially.

pause = False

# Loads the play button that shows up on the pause menu, which is used to resume
the game.

play_btn = IconButton(Image.PLAY_IMAGE)
```

```
# Game screen configuration begins below.
# Below three variables are for the pause function, hold the user lives
information and velocity of the lasers fired.
# In case the user selects the keyboard option, isMouse is set to false. Else, it
is set to true in the function call.

def game(isMouse=False):
    global pause
    lives = 5
    laser_vel = 10

    # Sets various parameters for the three fonts being loaded (such as the font
size).

    sub_font = pygame.font.Font(Font.neue_font, 40)
    sub_small_font = pygame.font.Font(Font.neue_font, 35)
    pop_up_font = pygame.font.Font(Font.edit_undo_font, 55)

    # Loads and plays the in-game music

    audio_cfg.play_music(Path.GAME_MUSIC_PATH)

    # Arrays that holds details of the enemy type to spawn, their velocity and
length of each level.

    enemies = []
    wave_length = 0
    enemy_vel = 1

    # Hide the mouse if player uses it for controlling spaceship. If using
keyboard, display mouse.

    player = Player(config.center_x, 585, mouse_movement=isMouse)
    if isMouse == True:
        pygame.mouse.set_visible(False)
    elif isMouse == False:
        pygame.mouse.set_visible(True)

    # Boolean variables that trigger victory, defeat or the boss level.

    lost = False
    win = False
    boss_entry = True
```

```
# Loads the pause button image.

    pause_btn = IconButton(Image.PAUSE_IMAGE)

    # Removes all pygame sprites from the explosion group.

    explosion_group.empty()

    # Re-draw window function below is used to return to gameplay window in-case
any external event is called that changes the same.

    def redraw_window():
        bg_obj.update()
        bg_obj.render()

        # Draws the player's ship on-screen using graphics from assets folder.

        player.draw()

        # Draws the enemy spaceships on-screen corresponding to the spaceship
name stored in the array defined above.

        for enemyShip in enemies:
            enemyShip.draw()

        # If the game is paused, draw the play button at (y=45). If the game is
not paused, draw the pause button at (y=45)

        if pause == True:
            play_btn.draw((config.center_x, 45), True, True)
        else:
            pause_btn.draw((config.center_x, 45), True, True)

        # Shows the number of lives the user has by drawing hearts using graphics
from assets folder.
        # (Default = 5 lives)

        for index in range(1, lives + 1):
            Assets.image.draw(Image.HEART_IMAGE,
                              (config.starting_x + 37 * index - 7, 30))
```

```
# Draw the current level number in the too-left corner, right below the number of
lives (hearts).

        Assets.text.draw(f'{player.get_level()} / 10', sub_small_font,
Colors.CYAN,
                         (config.starting_x + 33, 75))

        # Used to move the 'star' icon next to score a bit to the left.
        # This happens once when the score count crosses 100 and then again when
it crosses 1000.

        score = player.get_score()
        leftScoreIdx = 0
        if score >= 100 and score < 1000:
            leftScoreIdx = 1
        elif score >= 1000:
            leftScoreIdx = 2

        # Renders & draws the score values as text in green color on the screen,
along with the star icon.

        score_label = Assets.text.render(
            f'{score}', sub_font, Colors.GREEN)
        Assets.text.drawSurface(
            score_label, (config.ending_x - score_label.get_width() - 30, 20))
        Assets.image.draw(Image.STAR_IMAGE,
                          (config.ending_x - Image.SKULL_IMAGE.get_width() - 85 -
leftScoreIdx*23, 26))

        # Used to move the 'skull' icon next to kill count a bit to the left.
        # This happens once when the score count crosses 100.

        kills = player.get_kills()
        leftKillsIdx = 0
        if kills >= 100:
            leftKillsIdx = 1
```

```
        # Renders & draws the skull icon along with the kill count as text in red
on the screen.

        Assets.image.draw(Image.SKULL_IMAGE,
                          (config.ending_x - Image.SKULL_IMAGE.get_width() - 85 -
leftKillsIdx*15, 82))
        kills_label = Assets.text.render(
            f'{kills}', sub_font, Colors.RED)
        Assets.text.drawSurface(
            kills_label, (config.ending_x - kills_label.get_width() - 30, 75))


        # Display the text 'WINNER :)' if the player clears 10 levels and defeats
the boss too.

        if win:
            scores.append(True, player.get_level(), player.get_score(),
player.get_kills())
            Assets.text.draw('WINNER :)', pop_up_font, Colors.GREEN,
                             (config.center_x, 350), True)

        # Display the text 'GAME OVER :)' if the player uses up all lives or
allows too many enemies to pass through.

        if lost:
            scores.append(False, player.get_level(), player.get_score(),
player.get_kills())
            Assets.text.draw('GAME OVER :(', pop_up_font, Colors.RED,
                             (config.center_x, 350), True)

        # Display the text 'BOSS LEVEL!!' if the player clears 10 levels.

        if player.get_level() >= 10 and boss_entry:
            Assets.text.draw('BOSS LEVEL!!', pop_up_font, Colors.RED,
                             (config.center_x, 350), True)

        # Used to draw and update the explosions when player's laser hits an
enemy spaceship or vice-versa.

        explosion_group.draw(config.CANVAS)
        explosion_group.update()
```

```
        # Code to display the volume information.

        audio_cfg.display_volume()

        # Code to set the framerate to 60 FPS.

        pygame.display.flip()
        config.clock.tick(config.FPS)

    # Below codes contains a bit of game logic.
    # This includes generating enemies randomly on each level.
    # Each enemy spaceship has a codename - easy, medium, hard and boss.
    # There's also a level length which determines the number of enemies that
will be generated (less on lower levels).
    # The requisite function determines the level length and generates the
enemies.
    # When enemy count reaches zero, it advances to next level.

    while player.run:
        redraw_window()

        # When health bar reaches zero, deduct a life and refil the health bar.
        # If the lives reach zero, the 'lost' value is set to 'true' to signal
defeat.

        if lives > 0:
            if player.health <= 0:
                lives -= 1
                player.health = 100
        else:
            lost = True
            redraw_window()
            time.sleep(3)
            player.run = False
            pygame.mouse.set_visible(True)
```

```
# If player crosses level 10, entry to the boss level is signalled.
        # If player manages to defeat the boss, the 'win' value is set to 'true'
to signal victory.

        if player.get_level() == 10 and boss_entry:
            redraw_window()
            time.sleep(2)
            boss_entry = False
        elif player.get_level() > 10:
            win = True
            redraw_window()
            time.sleep(3)
            player.run = False

        # Enemy generation, level length determination and level system function
as explained at the top.

        if len(enemies) == 0:
            player.set_level()
            wave_length += 4

            for i in range(wave_length if player.get_level() < 10 else 1):
                enemies.append(Enemy(
                    random.randrange(50, config.WIDTH - 100),
                    random.randrange(-1200, -100),
                    random.choice(['easy', 'medium', 'hard']) if
player.get_level() < 10 else 'boss')
                )

        # Code to quit the game in case the user presses the 'X' button on
window.

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit(0)

        # Code to update the window size and dimensions of the background image
in the event user resizes the window.

            if event.type == pygame.VIDEORESIZE:
                if not display_cfg.fullscreen:
                    config.update(event.w, event.h)
```

```
        # Code to respond to mouse click events on buttons and in this case
specifically, the pause command.

            if event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                    if pause_btn.isOver():
                        pygame.mouse.set_visible(True)
                        pause = True
                        redraw_window()
                        paused(player, isMouse)

            # Code to implement various keyboard button functions like modifying
volume, quit game, mute, toggling full screen.
            # Also containts code that pauses the game when 'P' key is pressed.

            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_m:
                    audio_cfg.toggle_mute()
                if event.key == pygame.K_PLUS or event.key == pygame.K_EQUALS:
                    audio_cfg.inc_volume(5)
                if event.key == pygame.K_MINUS:
                    audio_cfg.dec_volume(5)
                if event.key == pygame.K_f:
                    config.update(
                        config.monitor_size[0], config.monitor_size[1])
                    display_cfg.toggle_full_screen()
                if event.key == pygame.K_p:
                    pygame.mouse.set_visible(True)
                    pause = True
                    redraw_window()
                    paused(player, isMouse)

        # Moves the player sprite.

        player.move()

        # Implements logic for firing enemy laser weapon system. A set laster
velocity is used while range is randomly generated.

        for enemy in enemies[:]:
            enemy.move(enemy_vel)
            enemy.move_lasers(laser_vel, player)

            if random.randrange(0, 2 * config.FPS) == 1:
                enemy.shoot()
```

```
# Implements the logic to increase kills & score while reducing health when the
player's ship collides with an enemy ship.

            if collide(enemy, player):
                player.SCORE += 50
                player.KILLS += 1
                if enemy.ship_type == 'boss':
                    if enemy.boss_max_health - 5 <= 0:

                        # If the player is able to defeat the boss, it will
trigger the boss explosion cutscene, followed by victory.

                        # Note: This is not seen as game is paused as soon as
boss health reaches zero.
                        # It is on our TODO list to be fixed. (probably implement
a short delay in pausing).

                        # Boss's laser weapons are capable of destroying player's
ship in single shot as damage value is 100.

                        boss_crash = Explosion(player.x, player.y, size=100)
                        explosion_group.add(boss_crash)

                        enemies.remove(enemy)
                        enemy.boss_max_health = 100
                        player.health -= 100
                    else:
                        enemy.boss_max_health -= 5
                        player.health -= 100

                        # Triggers the player death explosion.

                        crash = Explosion(player.x, player.y)
                        explosion_group.add(crash)
                else:
                    player.health -= 10
                    crash = Explosion(enemy.x, enemy.y)
                    explosion_group.add(crash)
                    enemies.remove(enemy)
            elif enemy.y + enemy.get_height()/2 > config.HEIGHT:
                lives -= 1
                enemies.remove(enemy)

        player.move_lasers(-laser_vel, enemies)
```

```
# Code to implement game pause is given below.
# Game pauses when 'P' key is pressed or when the 'Pause' button on the top is
pressed with mouse.
# Player can choose to continue the game or return to the main menu.

def paused(player, isMouse):

    # Loads the font that will be used to display text

    main_font = pygame.font.Font(Font.edit_undo_font, 60)

    # Renders and displays the text 'Paused' on the screen in cyan color.

    pause_label = Assets.text.render('Paused', main_font, Colors.CYAN)
    Assets.text.drawSurface(
        pause_label, (config.center_x - pause_label.get_width()//2, 300))

    # Loads the 'home' and 'play' button that is displayed when the game is
paused.

    play_2_btn = IconButton(Image.PLAY_IMAGE_2)
    home_btn = IconButton(Image.HOME_IMAGE)

    while pause:

        # Draws the 'home' and 'play' button at (x+66, 400) & (x-84, 400)
respectively.

        home_btn.draw((config.center_x+66, 400), True, True)
        play_2_btn.draw((config.center_x-84, 400), True, True)

        # Code to update the window size and dimensions of the background image
in the event user resizes the window.

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
```

```
# Code to respond to mouse click events on buttons. If user right clicks, it
returns to main menu.
            # If user clicks 'play' button, un-pause the game.
            # If user clicks 'home' button, return back to the main menu.

             if event.type == pygame.MOUSEBUTTONDOWN:
                 if event.button == 1:
                     if play_btn.isOver():
                         if isMouse == True:
                             pygame.mouse.set_visible(False)
                         elif isMouse == False:
                             pygame.mouse.set_visible(True)
                         unpause()
                     if play_2_btn.isOver():
                         if isMouse == True:
                             pygame.mouse.set_visible(False)
                         elif isMouse == False:
                             pygame.mouse.set_visible(True)
                         unpause()
                     if home_btn.isOver():
                         scores.append(False, player.get_level(),
player.get_score(), player.get_kills())
                         player.run = False
                         unpause()
                         audio_cfg.play_music(Path.MENU_MUSIC_PATH)

             # Code to implement various keyboard button functions.
             # When 'P' is pressed, the game is paused.
             # When 'backspace is pressed, the game returns to main menu.
             # Also contains code to disable mouse when using it for controlling
the spaceship and vice-versa.

             if event.type == pygame.KEYDOWN:
                 if event.key == pygame.K_p:
                     if isMouse == True:
                         pygame.mouse.set_visible(False)
                     elif isMouse == False:
                         pygame.mouse.set_visible(True)
                     unpause()
                 if event.key == pygame.K_BACKSPACE:
                     scores.append(False, player.get_level(), player.get_score(),
player.get_kills())
                     player.run = False
                     unpause()
```

```
# Plays the main menu music.

                    audio_cfg.play_music(Path.MENU_MUSIC_PATH)

        pygame.display.flip()
        config.clock.tick(15)

# Global unpause variable that sets value of 'pause' to 'false'.
def unpause():
    global pause
    pause = False
```

```
# Filename: ./screens/score_board.py

# Function: This file contains the code required to display the 'Scoreboard' page
in the game.

# The scoreboard contains the scores of every turns played in the game with the
number of kills & levels completed.

# Importing all modules required for proper functioning of the code in this file.

import pygame
import sys

# Importing the required code from other modules of the game.

from .background import slow_bg_obj
from models.icon_button import IconButton
from models.controls import audio_cfg, display_cfg
from models.scores import scores
from utils.assets import Assets
from config import config
from constants import Image, Font, Text, Colors

# Scoreboard page configuration data begins below.
# First one specifices & loads the fonts to be used (which are Edit Undo & Neue
Sans).

def score_board():
    score_title_font = pygame.font.Font(Font.edit_undo_font, 50)
    score_font = pygame.font.Font(Font.neue_font, 35)

    # Loads the image for the back button shown on the page.

    go_back_btn = IconButton(Image.GO_BACK_IMAGE)

    # Loop that runs the current screen. Value of run changes to false when game
begins, another screen is called or if user quits the game.
    # First command updates the background image (i.e. space) and renders it
dynamically.

    run = True
    while run:
        slow_bg_obj.update()
        slow_bg_obj.render()
```

```
# Draws the text 'SCOREBOARD' in green color on top of the screen.
        # Also draws a trophy cup on the right side of text.

        Assets.text.draw(Text.SCOREBOARD, score_title_font, Colors.GREEN,
                         (config.center_x, 90), True, False, True)
        Assets.image.draw(Image.TROPHY_IMAGE, (config.center_x + 160, 90))

        # Draws the text 'You haven't played yet!' in white color at (y=180) if
the game is run afresh with no games played so far.

        if len(scores.get_scores()) == 0:
            Assets.text.draw('You haven\'t played yet!', score_font, Colors.CYAN,
                             (config.center_x, 180), True)

        else:

            # Draws the image for the levels, kills and score buttons using files
from the assets folder.

            Assets.image.draw(Image.LEVELS_IMAGE,
                              (config.center_x-105, 160), True)
            Assets.image.draw(Image.KILLS_IMAGE,
                              (config.center_x+52, 160), True)
            Assets.image.draw(Image.SCORE_IMAGE,
                              (config.center_x+222, 160), True)

            # If the user has completed all 10 levels and defeated the boss, the
trophy cup image is drawn. If not, a skull is drawn.

            for i, item in enumerate(scores.get_top_5()):
                if item['status']:
                    Assets.image.draw(
                        Image.WON_IMAGE, (config.center_x-245, 240 + i*100),
True, True)
                else:
                    Assets.image.draw(Image.SKULL_IMAGE_2,
                                      (config.center_x-245, 240 + i*100), True,
True)
```

```
# Draws the values of levels, kills & score for each turn in cyan, red and yellow
respectively.

                Assets.text.draw(str(item['level']), score_font, Colors.CYAN,
                                 (config.center_x-105, 220 + i*100), True)
                Assets.text.draw(str(item['kills']), score_font, Colors.RED,
                                 (config.center_x+52, 220 + i*100), True)
                Assets.text.draw(str(item['score']), score_font, Colors.YELLOW,
                                 (config.center_x+222, 220 + i*100), True)

        # Draws the back button used to go back to the main menu at (x+65, 50).

        go_back_btn.draw((config.starting_x + 65, 50), True, True)

        # The following code given below are present on every screen component to
ensure smooth functioning of the game.

        # They include:

        # (a) Code to display the volume information
        # (b) Code to set the framerate to 60 FPS.
        # (c) Code to quit the game in case the user presses the 'X' button on
window.
        # (d) Code to update the window size and dimensions of the background
image in the event user resizes the window.
        # (e) Code to implement various keyboard button functions like modifying
volume, quit game, mute, toggling full screen.
        # (f) Code to respond to mouse click events on buttons, set 'run' to
false and return to main menu.
        # (g) Code to respond to mouse hover events, set the white outline over
interactive objects.

        # Each code block will be highlighted with their corresponding alphabets.

        # (a)
        audio_cfg.display_volume()

        # (b)
        config.clock.tick(config.FPS)
        pygame.display.flip()
```

```
        # (c)
        for event in pygame.event.get():


            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit(0)

            # (d)
            if event.type == pygame.VIDEORESIZE:
                if not display_cfg.fullscreen:
                    config.update(event.w, event.h)

            # (e)
            if event.type == pygame.KEYUP:
                if event.key == pygame.K_m:
                    audio_cfg.toggle_mute()
                if event.key == pygame.K_PLUS or event.key == pygame.K_EQUALS:
                    audio_cfg.inc_volume(5)
                if event.key == pygame.K_MINUS:
                    audio_cfg.dec_volume(5)
                if event.key == pygame.K_f:
                    config.update(
                        config.monitor_size[0], config.monitor_size[1])
                    display_cfg.toggle_full_screen()
                if event.key == pygame.K_BACKSPACE:
                    run = False

            # (f)
            if event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                    if go_back_btn.isOver():
                        run = False

            # (g)
            if event.type == pygame.MOUSEMOTION:
                if go_back_btn.isOver():
                    go_back_btn.outline = True
                else:
                    go_back_btn.outline = False
```

```
# Filename: ./screens/settings.py

# Function: This file contains the code required to display the 'Settings'
page in the game.

# In the settings page, we only have one option to be modified, which is the
volume of the game.

# Importing all modules required for proper functioning of the code in this
file.

import pygame
import sys

# Importing the required code from other modules of the game.

from .background import slow_bg_obj
from models.icon_button import IconButton
from models.controls import audio_cfg, display_cfg
from utils.assets import Assets
from config import config
from constants import Image, Font, Colors, Text

# Settings page configuration data begins below.
# First one specifices & loads the fonts to be used (which is Edit Undo).

def settings():
    settings_title_font = pygame.font.Font(Font.edit_undo_font, 50)
    settings_right_font = pygame.font.Font(Font.edit_undo_font, 50)
    settings_left_font = pygame.font.Font(Font.edit_undo_font, 46)

    # Loads the image for the back button shown on the page.

    go_back_btn = IconButton(Image.GO_BACK_IMAGE)

    # Loads the images for the plus and minus button used to increase /
decrease volume.

    plus_btn = IconButton(Image.PLUS_IMAGE)
    minus_btn = IconButton(Image.MINUS_IMAGE)

    # Loop that runs the current screen. Value of run changes to false when
game begins, another screen is called or if user quits the game.
    # First command updates the background image (i.e. space) and renders it
dynamically.

    run = True
```

```
    while run:
        slow_bg_obj.update()
        slow_bg_obj.render()

        # Draws the text 'SETTINGS' in yellow color on top of the screen.
        # Also draws spanner & toolbox icons on the left & right of text
respectively.

        Assets.text.draw(Text.SETTINGS, settings_title_font, Colors.YELLOW,
                         (config.center_x, 130), True, False, True)
        Assets.image.draw(Image.TOOLS_IMAGE,
                          (config.center_x - 150, 120), True)
        Assets.image.draw(Image.TOOLBOX_IMAGE,
                          (config.center_x + 150, 129), True)

        # Draws the text 'VOLUME' with green color at (x-160, 240).
        # Also draws the volume value as numbers in white color at (x+155,
240).

        Assets.text.draw('VOLUME', settings_left_font, Colors.GREEN,
                         (config.center_x - 160, 240), True)
        Assets.text.draw(f'{audio_cfg.volume}', settings_right_font,
Colors.WHITE,
                         (config.center_x + 155, 240), True)

        # Draws the back button used to go back to the main menu at (x+65,
50).

        go_back_btn.draw((config.starting_x + 65, 50), True, True)

        # Draws the plus and minus buttons used to increase & decrease volumes
at (x+235, 230) & (x+70, 260) respectively.

        plus_btn.draw((config.center_x + 235, 260), True, True)
        minus_btn.draw((config.center_x + 70, 260), True, True)

        # The following code given below are present on every screen component
to ensure smooth functioning of the game.

        # They include:

        # (a) Code to set the framerate to 60 FPS.
        # (b) Code to quit the game in case the user presses the 'X' button on
window.
        # (c) Code to update the window size and dimensions of the background
image in the event user resizes the window.
        # (d) Code to implement various keyboard button functions like
modifying volume, quit game, mute, toggling full screen.
```

```
        # (e) Code to respond to mouse click events on buttons, set 'run' to
false and return to main menu.
        # (f) Code to respond to mouse hover events, set the white outline
over interactive objects.

        # Each code block will be highlighted with their corresponding
alphabets.

        # Note: Volume is not being displayed at the bottom left corner as in
other pages since we're modifying the volume here!

        # (a)
        config.clock.tick(config.FPS)
        pygame.display.flip()


        for event in pygame.event.get():

            # (b)
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit(0)

            # (c)
            if event.type == pygame.VIDEORESIZE:
                if not display_cfg.fullscreen:
                    config.update(event.w, event.h)

            # (d)
            if event.type == pygame.KEYUP:
                if event.key == pygame.K_f:
                    config.update(
                        config.monitor_size[0], config.monitor_size[1])
                    display_cfg.toggle_full_screen()
                if event.key == pygame.K_BACKSPACE:
                    run = False

            # (e)
            if event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                    if go_back_btn.isOver():
                        run = False
                    if plus_btn.isOver():
                        audio_cfg.inc_volume(5)
                    if minus_btn.isOver():
                        audio_cfg.dec_volume(5)
```

```
# (f)
            if event.type == pygame.MOUSEMOTION:
                if go_back_btn.isOver():
                    go_back_btn.outline = True
                else:
                    go_back_btn.outline = False
```

```
# Filename: ./screens/ships.py

# Function: This file contains the code required to display the 'Ships' page
in the game.

################################################################################

#There are 4 different enemy spaceships in the game. They are:

# 1. Lanius Outrider (Easy)
# Health: 100, Weapon Damage: 10

# 2. Zoltan Interceptor (Medium)
# Health: 100, Weapon Damage: 18

# 3. Slug Instigator (Hard)
# Health: 100, Weapon Damage: 25

# 4. Mantis Battlecruiser (Boss)
# Health: 1980, Weapon Damage: 100

# The user's spacecraft has the following attributes:
# Health: 100, Lives: 5, Weapon Damage: 100

################################################################################

# Importing all modules required for proper functioning of the code in this
file.


import pygame
import sys

# Importing the required code from other modules of the game.

from .background import slow_bg_obj
from models.icon_button import IconButton
from models.controls import audio_cfg, display_cfg
from utils.assets import Assets
from config import config
from constants import Image, Font, Colors, Text

# Ship page configuration data begins below.
# First one specifices & loads the fonts to be used (which is Edit Undo and
Neue Sans).

def ships():
```

```
    ships_title_font = pygame.font.Font(Font.edit_undo_font, 50)
    ships_info_font = pygame.font.Font(Font.neue_font, 22)

    # Loads the image for the back button shown on the page.

    go_back_btn = IconButton(Image.GO_BACK_IMAGE)

    # Loads the image for the heart, which depicts the number of the lives the
player's ship has.

    NEW_HEART_IMAGE = pygame.transform.scale(
        Image.HEART_IMAGE, (Image.HEART_IMAGE.get_width()*3/4,
Image.HEART_IMAGE.get_height()*3/4))

    # Loop that runs the current screen. Value of run changes to false when
game begins, another screen is called or if user quits the game.
    # First command updates the background image (i.e. space) and renders it
dynamically
    run = True
    while run:
        slow_bg_obj.update()
        slow_bg_obj.render()

        # Draws the text 'SHIPS' in cyan color on top of the screen with two
spaceship icons on either side (L & R) of the text.

        Assets.text.draw(Text.SHIPS, ships_title_font, Colors.CYAN,
                         (config.center_x, 100), True, False, True)
        Assets.image.draw(Image.SHIPS_IMAGE,
                          (config.center_x - 110, 90), True)
        Assets.image.draw(Image.SHIPS_IMAGE_2,
                          (config.center_x + 110, 99), True)

        # Codes given below draws the images of spaceships with their
information such as health, weapon damage & name.

        # Name: Lanius Outrider; Health: 100; Damage: 10;

        Assets.image.draw(Image.EASY_SPACE_SHIP,
                          (config.center_x - 270, 210), True)
        Assets.text.draw('Lanius Outrider', ships_info_font, Colors.WHITE,
                         (config.center_x - 210, 195))
        Assets.text.draw('Health: 100', ships_info_font,
                         Colors.GREEN, (config.center_x - 210, 222))
        Assets.text.draw('Damage: 10', ships_info_font,
                         Colors.RED, (config.center_x - 210, 249))

        # Name: Zoltan Interceptor; Health: 100; Damage: 18;
```

```
        Assets.image.draw(Image.MEDIUM_SPACE_SHIP,
                          (config.center_x - 270, 295), True)
        Assets.text.draw('Zoltan Interceptor', ships_info_font, Colors.WHITE,
                         (config.center_x - 210, 290))
        Assets.text.draw('Health: 100', ships_info_font,
                         Colors.GREEN, (config.center_x - 210, 317))
        Assets.text.draw('Damage: 18', ships_info_font,
                         Colors.RED, (config.center_x - 210, 344))

        # Name: Slug Instigator; Health: 100; Damage: 25;

        Assets.image.draw(Image.HARD_SPACE_SHIP,
                          (config.center_x - 270, 420), True)
        Assets.text.draw('Slug Instigator', ships_info_font, Colors.WHITE,
                         (config.center_x - 210, 400))
        Assets.text.draw('Health: 100', ships_info_font,
                         Colors.GREEN, (config.center_x - 210, 427))
        Assets.text.draw('Damage: 25', ships_info_font,
                         Colors.RED, (config.center_x - 210, 454))

        # Name: Your Spaceship; Lives: 5; Health: 100; Damage: 100

        Assets.image.draw(Image.PLAYER_SPACE_SHIP,
                          (config.center_x + 260, 270), True)
        Assets.text.draw('Your Spaceship', ships_info_font, Colors.WHITE,
                         (config.center_x + 70, 265))
        for index in range(1, 6):
            Assets.image.draw(
                NEW_HEART_IMAGE, (config.center_x + 45 + 25 * index, 292))
        Assets.text.draw('Health: 100', ships_info_font,
                         Colors.GREEN, (config.center_x + 70, 319))
        Assets.text.draw('Damage: 100', ships_info_font,
                         Colors.RED, (config.center_x + 70, 346))

        # Name: Mantis Battlecruiser; Health: 1980; Damage: 100;

        Assets.image.draw(Image.BOSS_SHIP, (config.center_x, 450), True)
        Assets.text.draw('Mantis Battlecruiser', ships_info_font,
Colors.WHITE,
                         (config.center_x + 150, 505))
        Assets.text.draw('Health: 1980', ships_info_font,
                         Colors.GREEN, (config.center_x + 150, 532))
        Assets.text.draw('Damage: 100', ships_info_font,
                         Colors.RED, (config.center_x + 150, 559))

        # Draws the back button used to go back to the main menu at (x+65,
50).
```

```
        go_back_btn.draw((config.starting_x + 65, 50), True, True)

        # The following code given below are present on every screen component
to ensure smooth functioning of the game.

        # They include:

        # (a) Code to display the volume information
        # (b) Code to set the framerate to 60 FPS.
        # (c) Code to quit the game in case the user presses the 'X' button on
window.
        # (d) Code to update the window size and dimensions of the background
image in the event user resizes the window.
        # (e) Code to implement various keyboard button functions like
modifying volume, quit game, mute, toggling full screen.
        # (f) Code to respond to mouse click events on buttons, set 'run' to
false and return to main menu.
        # (g) Code to respond to mouse hover events, set the white outline
over interactive objects.

        # Each code block will be highlighted with their corresponding
alphabets.

        # (a)
        audio_cfg.display_volume()

        # (b)
        config.clock.tick(config.FPS)
        pygame.display.flip()


        for event in pygame.event.get():

            # (c)
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit(0)

            # (d)
            if event.type == pygame.VIDEORESIZE:
                if not display_cfg.fullscreen:
                    config.update(event.w, event.h)

            # (e)
            if event.type == pygame.KEYUP:
                if event.key == pygame.K_m:
                    audio_cfg.toggle_mute()
```

```
            if event.key == pygame.K_PLUS or event.key == pygame.K_EQUALS:
                audio_cfg.inc_volume(5)
            if event.key == pygame.K_MINUS:
                audio_cfg.dec_volume(5)
            if event.key == pygame.K_f:
                config.update(
                    config.monitor_size[0], config.monitor_size[1])
                display_cfg.toggle_full_screen()
            if event.key == pygame.K_BACKSPACE:
                run = False

        # (f)
        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1:
                if go_back_btn.isOver():
                    run = False

        # (g)
        if event.type == pygame.MOUSEMOTION:
            if go_back_btn.isOver():
                go_back_btn.outline = True
            else:
                go_back_btn.outline = False
```

```
# Filename: ./utils/assets.py

# Function: Utility file that defines various functions used by asset files,
including rendering, drawing and scaling..

# Importing all modules required for proper functioning of the code in this
file.

import pygame
import os

# Importing the required code from other modules of the game.

from utils.resource_path import resource_path
from config import config

# Defines various functions used by graphic/audio assets.

class Assets:
    class text:

        # The below function is used to render text on the screen. Inputs to
be provided are the text, font name and color.

        def render(text, font, color):
            return font.render(text, 1, color)

        # The below function is used to draw text on screen with position.
Inputs to be provided are the text, font name and color.
        # Position can be specified using coorindates or set (x,y) values from
the center using isCenterX and isCenterY.

        def draw(text, font, color, pos, isCenterX=False, isCenterY=False,
underline=False):
            text_label = font.render(text, 1, color)

            # Defines how to calculate position if isCenterX and isCenterY is
provided as input.

            if isCenterX:
                pos = (pos[0] - text_label.get_width()//2, pos[1])
            if isCenterY:
                pos = (pos[0], pos[1] - text_label.get_height()//2)

            if underline:

                # Draws a straight line
```

```
                pygame.draw.line(config.CANVAS, color,
                                 (pos[0], pos[1]+45),
(pos[0]+text_label.get_width()-4, pos[1]+45), 7)

            config.CANVAS.blit(text_label, pos)

        # Used to draw a surface with a given label at a given position.

        def drawSurface(label, pos):
            config.CANVAS.blit(label, pos)

    # Various image related functions are defined below.

    class image:

        # Used to load an image file. The root folder containing the file can
be specified followed by the complete image path.
        # Returns the resource path to the image

        def load(root_path, image_path):
            return pygame.image.load(resource_path(os.path.join(root_path,
image_path)))

        # Used to load an image file and then scale it on the fly
(dynamically).
        # The root folder containing the file can be specified followed by the
complete image path and the scale factor.
        # Returns the scaled image.

        def scale(root_path, image_path, factor):
            image = pygame.image.load(resource_path(
                os.path.join(root_path, image_path)))

            return pygame.transform.scale(image, (image.get_width()*factor,
image.get_height()*factor))

        # Used to draw image on the screen at a given position.
        # Position can be specified using coorindates or set (x,y) values from
the center using isCenterX and isCenterY.

        def draw(image, pos, isCenterX=False, isCenterY=False):
            if isCenterX:
                pos = (pos[0] - image.get_width()//2, pos[1])
            if isCenterY:
                pos = (pos[0], pos[1] - image.get_height()//2)

            config.CANVAS.blit(image, pos)
```

```
    class sound:

        # Used to load audio files required by the game. The root folder
containing the audio can be specified followed by the complete path.
        # Returns the audio file. We use mixer, a component provided by pygame
to manage audio I/O.

        def load(root_path, sound_path):
            return pygame.mixer.Sound(resource_path(os.path.join(root_path,
sound_path)))

    class font:

        # Used to load fonts required by the game. The root folder containing
the font can be specified followed by the complete path.

        def load(root_path, font_path):
            return resource_path(os.path.join(root_path, font_path))
```

```
# Filename: ./utils/collide.py

# Function: Utility file that contains the code to generate collision masks
for spaceships.

# Essentially, we need a way to detect when the player's spaceship collides
into an enemy spaceship.
# To detect the same, we use this function. The function creates a mask, just
like in the outline.

# This mask data is returned, which is then used to detect collision and
adjust the gameplay value accordingly.

def collide(obj1, obj2):
    # obj1 and obj2 coordinates refer to the middle point of the mask, so we
have to compute
    # the coordinates of the upper-left corner of the sprite.
    x_offset = int((obj2.x - obj2.get_width()/2) -
                   (obj1.x - obj1.get_width()/2))
    y_offset = int((obj2.y - obj2.get_height()/2) -
                   (obj1.y - obj1.get_height()/2))
    return obj1.mask.overlap(obj2.mask, (x_offset, y_offset)) != None
```

```python
# Filename: ./utils/outlineImage.py

# Function: Utility file that contains the code to create the while outlines
on any interactive object the mouse hovers on.

# Importing all modules required for proper functioning of the code in this
file.

import pygame

# Importing the required code from other modules of the game.

from constants import Colors
from utils.assets import Assets

# Outline definition given below.
# Essentially, the code works by providing an input image along with it's
position.
# A mask is then generated for the image/button in question.
# We want our outline to be white in color, hence we specify the argument
accordingly.
# Finally, we use the draw function to draw the outline at the border of this
mask.
# This shows up as the outline on interactive objects like buttons.

def outlineImage(image, pos):
    mask = pygame.mask.from_surface(image)
    mask_outline = mask.outline()
    mask_surf = pygame.Surface(image.get_size())
    for pixel in mask_outline:
        mask_surf.set_at(pixel, Colors.WHITE)
    mask_surf.set_colorkey((0, 0, 0))

    Assets.image.draw(mask_surf, (pos[0], pos[1]+2))
    Assets.image.draw(mask_surf, (pos[0], pos[1]+1))
    Assets.image.draw(mask_surf, (pos[0], pos[1]-1))
    Assets.image.draw(mask_surf, (pos[0], pos[1]-2))
    Assets.image.draw(mask_surf, (pos[0]+2, pos[1]))
    Assets.image.draw(mask_surf, (pos[0]+1, pos[1]))
    Assets.image.draw(mask_surf, (pos[0]-1, pos[1]))
    Assets.image.draw(mask_surf, (pos[0]-2, pos[1]))
    Assets.image.draw(mask_surf, (pos[0]+1, pos[1]+1))
    Assets.image.draw(mask_surf, (pos[0]+1, pos[1]-1))
    Assets.image.draw(mask_surf, (pos[0]-1, pos[1]+1))
    Assets.image.draw(mask_surf, (pos[0]-1, pos[1]-1))
```

```
# Filename: ./utils/resource_path.py

# Function: Utility file that defines the resource path and how to open files
from a base path when provided.

# Importing all modules required for proper functioning of the code in this
file.

import os
import sys

# Resource path information defined below.

def resource_path(relative_path):
    try:
        base_path = sys._MEIPASS # Stores the absolute (base) path to the
assets in sys._MEIPASS.
    except Exception:
        base_path = os.path.abspath(".") # Returns a normalized version of the
pathname "."

    return os.path.join(base_path, relative_path) # Returns a string which
represents the concatenated path.
```

```
# Filename: ./models/button.py

# Function: Model file that contains various functions that are used by
buttons in-game. This include draw and initialization.

# Importing all modules required for proper functioning of the code in this
file.

import pygame

# Importing the required code from other modules of the game.

from utils.assets import Assets
from config import config
from constants import Font, Colors


class Button:

    # Initializes various button parameters. Input parameters include color,
outline color and text to be embedded in the button.

    def __init__(self, color, outline_color, text=''):
        self.color = color
        self.outline_color = outline_color
        self.text = text
        self.outline = False
        self.rect = pygame.Rect(0, 0, 0, 0)

    # Function that draws the button in question. Input include position and
size of the button.
    # Once the parameters are passed, the coordinates to draw are calculated
and if outline is required or not.
    # Finally, the button is drawn using draw.rect()

    def draw(self, pos, size):
        self.default_outline = pygame.Rect(
            pos[0]-5, pos[1]-5, size[0]+10, size[1]+10)
        self.on_over_outline = pygame.Rect(
            pos[0]-6, pos[1]-6, size[0]+12, size[1]+12)
        self.rect = self.default_outline

        # Default values to use incase no values are specified.
        default_inner_rect = (pos[0], pos[1], size[0], size[1])


        onover_inner_rect = (pos[0]+1, pos[1]+1, size[0]-2, size[1]-2)
        inner_rect = onover_inner_rect if self.outline == True else
default_inner_rect
```

```
        pygame.draw.rect(config.CANVAS, self.outline_color,
                         self.on_over_outline if self.outline == True else
self.default_outline, 0, 7)

        pygame.draw.rect(config.CANVAS, self.color, inner_rect, 0, 6)

        # After the button is drawn, if there any text to be embedded in the
box, it is performed by below code snippet.
        # Initializes the font required, color and position to draw.

        if self.text != '':
            font = pygame.font.Font(Font.neue_font, 40)
            Assets.text.draw(self.text, font, Colors.WHITE,
                             (pos[0] + size[0]/2, pos[1] + size[1]/2), True,
True)

     # Returns 'true' if that point is within the bounds of the rectangle.
Usually used to implement highlighting of button.

    def isOver(self):
        return self.rect.collidepoint(pygame.mouse.get_pos())
```

```
# Filename: ./models/controls.py

# Function: Model file that contains various audio / display related control
functions that the game uses.

# This includes adjusting the volume, playing the music, and dynamically
scaling to full-screen from windowed and vice-versa.

# Importing all modules required for proper functioning of the code in this
file.

import pygame

# Importing the required code from other modules of the game.

from utils.assets import Assets
from config import config
from constants import Image, soundList, Font, Colors

# Audio related control functions defined below.

class AudioControls:

    # Initializes audio settings to default values.

    def __init__(self, soundList):
        self.soundList = soundList
        self.volume = 100
        self.muted = True if self.volume == 0 else False
        self.prev_volume = -1


        pygame.mixer.music.set_volume(self.volume / 100)
        for soundItem in self.soundList:
            soundItem.set_volume(self.volume / 100)

    # Below function is used to set volume level. If it is 0, audio will be
disabled.
    # If it is a value greater than 0, mute value is set to false.
    def set_volume(self, level):
        if level == 0:
            self.muted = True
        if self.muted and level > 0:
            self.muted = False
            self.prev_volume = 50  # If you unmute at zero volume, volume
level defaults to 50.

        # Volume level is usually set in the range of 0 to 1 for pygame mixer.
```

```
        # Hence, we're converting the level in [0,1] by dividing it with 100.

        self.volume = level
        pygame.mixer.music.set_volume(level / 100)

        # Can be used to dynamically set different volumes for various sound
effects.

        for soundItem in soundList:
            soundItem.set_volume(level / 100)

    # Function that decreases volume in steps on the settings page.

    def dec_volume(self, amt):
        amt = max(0, self.volume - amt)
        self.set_volume(amt)

    # Function that increases volume in steps on the settings page.

    def inc_volume(self, amt):
        amt = min(100, self.volume + amt)
        self.set_volume(amt)

    # Function that decreases volume in steps on the settings page.
    def toggle_mute(self):
        if self.muted:
            self.set_volume(self.prev_volume)
        else:
            self.prev_volume = self.volume
            self.set_volume(0)

    # Function that displays the current volume level.

    def display_volume(self):
        control_font = pygame.font.Font(Font.neue_font, 30)

        # If muted, draw the mute icon. If not muted, draw the volume icon
accordingly.
        if self.muted:
            Assets.image.draw(
                Image.MUTE_ICON, (config.starting_x+20, config.ending_y-52))
            vol_lbl_text = " --"
        else:
            Assets.image.draw(
                Image.VOL_ICON, (config.starting_x + 20, config.ending_y -
52))
            vol_lbl_text = str(self.volume).rjust(3, " ")
```

```
        Assets.text.draw(vol_lbl_text, control_font, Colors.WHITE,
                         (config.starting_x + 70, config.ending_y - 57))

    # Function used to play music by loading the path.

    def play_music(self, path):
        pygame.mixer.music.load(path)
        pygame.mixer.music.play(-1)

# Display related functions defined below.

class DisplayControls:

    # Initializes default value (i.e. not full screen).

    def __init__(self):
        self.fullscreen = False

    # Function used to toggle full screen mode, upon which all elements are
dynamically scaled to fit the new resized window.
    # If not full screen mode, set window as resizable.

    def toggle_full_screen(self):
        self.fullscreen = not self.fullscreen
        if self.fullscreen:
            config.CANVAS = pygame.display.set_mode(
                config.monitor_size, pygame.FULLSCREEN)
        else:
            config.CANVAS = pygame.display.set_mode(
                (config.WIDTH, config.HEIGHT), pygame.RESIZABLE)


audio_cfg = AudioControls(soundList)
display_cfg = DisplayControls()
```

```
# Filename: ./models/explosion.py

# Function: Model file that defines how explosions work in-game.

# Importing all modules required for proper functioning of the code in this
file.

import os
import pygame

# Importing the required code from other modules of the game.

from utils.assets import Assets
from constants import Path, Sound

# Essentially, when it comes to implementing explosions, we have 7 PNG files.
These PNG files depict various phases of an explosion.
# When played in a continous fashion, it turns out to be an explosion.
# Initially, we went with a GIF but it had more memory and processing power
requirements. Hence the switch.

# Explosion sprite group definition.

explosion_group = pygame.sprite.Group()

# Explosion logic defined below.

class Explosion(pygame.sprite.Sprite):

    # Initializes default values (i.e. number of frames, explosion size)
    # All images are read and stored in array 'images[]'

    def __init__(self, x, y, size=60, num_frames=8):
        super().__init__()
        self.images = []

        # Play from tile1 to tile7 in quick succession.

        for num in range(0, num_frames):
            img = Assets.image.load(Path.EXPLOSION_PATH, f"tile{num:03}.png")
            img = pygame.transform.scale(img, (size, size))
            self.images.append(img)
        self.index = 0
        self.image = self.images[self.index]
        self.rect = self.image.get_rect()
        self.rect.center = [x, y]
        self.counter = 0
```

```
        # Plays different sound based on the explosion size (in this case, if
size < 40, play different audio).

        if size < 40:
            Sound.LASER_HIT_SOUND.play()
        else:
            Sound.EXPLODE_SOUND.play()

    def update(self):
        explosion_speed = 4

        # Update explosion animation by incrementing counter.

        self.counter += 1

        if self.counter >= explosion_speed and self.index < len(self.images) -
1:
            self.counter = 0
            self.index += 1
            self.image = self.images[self.index]

        # If the animation is complete, reset animation index for playing
again next time an explosion occurs.

        if self.index >= len(self.images) - 1 and self.counter >=
explosion_speed:
            self.kill()
```

```
# Filename: ./models/icon_button.py

# Function: Model file that contains various functions that are used by icon
type buttons in-game. This include draw and initialization.

# Importing all modules required for proper functioning of the code in this
file.

import pygame

# Importing the required code from other modules of the game.

from constants import Colors, Font
from utils.outlineImage import outlineImage
from utils.assets import Assets

class IconButton:

    # Initializes various button parameters. Input parameters include the
image file and text to be included, if any.

    def __init__(self, image, subtitle=''):
        self.image = image
        self.outline = False
        self.subtitle = subtitle
        self.rect = pygame.Rect(
            0, 0, self.image.get_width(), self.image.get_height())

    # Function that draws the button in question. Input include position and
size of the button.
    # Once the parameters are passed, the coordinates to draw are calculated
and if outline is required or not.
    # Note: isCenterX and isCenterY can also be used to specify coordinates.


    def draw(self, pos, isCenterX=False, isCenterY=False):
        new_pos = pos
        if isCenterX == True:
            new_pos = (new_pos[0] - self.image.get_width()//2, new_pos[1])
        if isCenterY == True:
            new_pos = (new_pos[0], new_pos[1] - self.image.get_height()//2)

        # If an outline is required for the icon, then values can be passed
accordingly.

        if self.outline == True:
            outlineImage(self.image, new_pos)
```

```
        self.rect = pygame.Rect(
            new_pos[0], new_pos[1], self.image.get_width(),
self.image.get_height())

        # Finally, the button is drawn using image.draw()

        Assets.image.draw(self.image, self.rect)

        # Specifices the subtitle font to be used, if there is text present.

        subtitle_font = pygame.font.Font(Font.neue_font, 20)

        # Draws the subtitle text in white color.

        if self.subtitle != '':
            Assets.text.draw(self.subtitle, subtitle_font, Colors.WHITE,
                             (pos[0], pos[1] + 35), True)

    # Returns 'true' if that point is within the bounds of the rectangle.
Usually used to implement highlighting of textbox.

    def isOver(self):
        return self.rect.collidepoint(pygame.mouse.get_pos())
```

```
# Filename: ./models/laser.py

# Function: Model file that contains various laser related functions that
essentially implement the weapon system.

# Importing all modules required for proper functioning of the code in this
file.

import pygame

# Importing the required code from other modules of the game.

from config import config
from utils.collide import collide
from utils.assets import Assets


class Laser:

    # Initializes the laser system. Inputs such as laser image (for player /
enemy) and coorindates can be provided.

    def __init__(self, x, y, img):
        self.x = x
        self.y = y
        self.img = img
        self.mask = pygame.mask.from_surface(self.img)

    # Draws the laser on the screen.

    def draw(self):

        # Makes laser's coordinates centered in the sprite.

        Assets.image.draw(
            self.img, (config.starting_x + self.x, self.y), True, True)

    # Propagates the laser across the screen with the specified velocity.

    def move(self, vel):
        self.y += vel

    # Defines what happens to the laser when it goes off-screen.

    def off_screen(self, height):
        return not(height >= self.y >= 0)
```

```
    # Defines what happens to the laser when it collides with the player.
Essentially, it reduces the player health acc. to laser type.

    def collision(self, obj):
        return collide(self, obj)

    # Defines the height and width of the laser beam.

    def get_width(self):
        return self.img.get_width()

    def get_height(self):
        return self.img.get_height()
```

```
# Filename: ./models/scores.py

# Function: Model file that contains functions used by the scoreboard.

# This include getting top 5 high scores (which are stored in an array),
displaying the list and even appending the values for a particular gameplay.

class Scores:

    # Initializes empty matrix to store score list.

    def __init__(self):
        self.score_list = []

    # Used to get top 5 high scores.

    def get_top_5(self):
        return sorted(self.score_list, key=lambda item: item['score'],
reverse=True)

    # Used to get complete list of high scores.

    def get_scores(self):
        return self.score_list

    # Appends values of level completed, kills made, score achieved for a
particular gameplay into matrix.

    def append(self, status, level, score, kills):
        self.score_list.append({
            "status": status,
            "level": level,
            "score": score,
            "kills": kills,
        })


scores = Scores()
```

```
# Filename: ./models/ship.py

# Function: Model that contains all functions, parameters and implementations
used by the spaceships.

# Importing all modules required for proper functioning of the code in this
file.

import pygame

# Importing the required code from other modules of the game.

from utils.assets import Assets
from models.laser import Laser
from models.explosion import Explosion, explosion_group
from models.controls import audio_cfg
from models.scores import scores
from config import config
from constants import Path, Image, Colors, Sound


class Ship:

    # Initializing required variables that enemy spaceship would use (incl.
position, health, ship image file and so on).

    def __init__(self, x, y, health=100):
        self.x = x
        self.y = y
        self.health = health
        self.ship_img = None
        self.laser_img = None
        self.lasers = []
        self.cool_down_counter = 0
        self.CoolDown = 25
        self.boss_max_health = 99
        self.SCORE = 0
        self.KILLS = 0
        self.level = 0

    def draw(self):

        # Draws lasers before the ship so that it doesn't appear like the
lasers appear from above the ship

        for laser in self.lasers:
            laser.draw()
```

```
        # Makes ship's coordinates centered in the sprite

        Assets.image.draw(
            self.ship_img, (config.starting_x+self.x, self.y), True, True)

    # This is responsible for propagating a fired laser through the screen.
Once it goes off-screen, it is removed.
    # If it collides with the player / enemy, it will destroy the enemy /
reduce health for player.

    def move_lasers(self, vel, obj):
        self.coolDown()
        for laser in self.lasers:
            laser.move(vel)
            if laser.off_screen(config.HEIGHT):
                self.lasers.remove(laser)
            elif laser.collision(obj):
                obj.health -= 10
                self.lasers.remove(laser)

    # Defines a cooldown period when enemy ships don't fire lasers.
    # When it expires, they start firing again.

    def coolDown(self):
        if self.cool_down_counter >= self.CoolDown:
            self.cool_down_counter = 0
        elif self.cool_down_counter > 0:
            self.cool_down_counter += 1

    # Responsible for firing lazer, when the cooldown timer value is set to 0.
    # Plays the laser firing audio and sends the laser image and position to
be drawn.

    def shoot(self):
        if self.cool_down_counter == 0:
            Sound.PLAYER_LASER_SOUND.play()
            laser = Laser(self.x, self.y, self.laser_img)
            self.lasers.append(laser)
            self.cool_down_counter = 1

    # Below two functions are used to get the width and height of the enemy
spaceship.

    def get_width(self):
        return self.ship_img.get_width()

    def get_height(self):
```

```
        return self.ship_img.get_height()

    # Below three functions return the current score, kill count and current
level.
    # They are usually saved in the scoreboard matrix file.

    def get_score(self):
        return self.SCORE

    def get_kills(self):
        return self.KILLS

    def get_level(self):
        return self.level

    # Used to go to the next level (essentially sets the next level) when the
current level is cleared.

    def set_level(self):
        self.level += 1

# Defines various variables related to the player's spaceship (incl. its
health, image, laser image, velocity and so on).
class Player(Ship):
    def __init__(self, x, y, health=100, mouse_movement=False):
        super().__init__(x, y, health)
        self.ship_img = Image.PLAYER_SPACE_SHIP
        self.laser_img = Image.PLAYER_LASER
        self.mask = pygame.mask.from_surface(self.ship_img)
        self.max_health = health
        self.mouse_movement = mouse_movement
        self.run = True
        self.vel = 5

    # Defines how to respond if the player chooses to use keyboard to play.
    # When the keys are pressed, it responds accordingly by calling the module
required.

    def move_with_keyboard(self):
        keys = pygame.key.get_pressed()
        action = {'LEFT': keys[pygame.K_LEFT] or keys[pygame.K_a],
                  'RIGHT': keys[pygame.K_RIGHT] or keys[pygame.K_d],
                  'UP': keys[pygame.K_UP] or keys[pygame.K_w],
                  'DOWN': keys[pygame.K_DOWN] or keys[pygame.K_s],
                  'SHOOT': keys[pygame.K_SPACE],
                  'QUIT': keys[pygame.K_BACKSPACE]}

        # Return to main page
```

```
        if action['QUIT']:
            audio_cfg.play_music(Path.MENU_MUSIC_PATH)
            self.run = False
        # Left Key
        if action['LEFT'] and (self.x - self.vel) > self.get_width()/2:
            self.x -= self.vel
        # Right Key
        if action['RIGHT'] and (self.x + self.vel + self.get_width()/2) <
config.WIDTH:
            self.x += self.vel
        # Up Key
        if action['UP'] and (self.y - self.vel) > 0:
            self.y -= self.vel
        # Down Key
        if action['DOWN'] and (self.y + self.vel + self.get_height()) <
config.HEIGHT:
            self.y += self.vel
        # Shoots Laser
        if action['SHOOT']:
            self.shoot()

    # Defines how to respond if the player chooses to use a mouse to play.

    def move_with_mouse(self):

        # Below variables store information like position of the mouse, when
it is pressed and when keys are also pressed.
        cx, cy = pygame.mouse.get_pos()
        button = pygame.mouse.get_pressed()
        keys = pygame.key.get_pressed()

        # Facilitates movement

        if cx > self.get_width()/2 and cx < config.WIDTH - self.get_width()/2
\
                and cy > 0 and cy < config.HEIGHT:
            self.x = cx
            self.y = cy

        # Shoots laser on left click of mouse or using spacebar.

        if button[0] or keys[pygame.K_SPACE]:
            self.shoot()

        # Returns to main menu if the right mouse button or backspace key is
pressed.
```

```
        if button[2] or keys[pygame.K_BACKSPACE]:
            score_obj = {
                "status": False,
                "level": self.get_level(),
                "score": self.get_score(),
                "kills": self.get_kills(),
            }

            # Saves the score data (which includes the levels finished, score
and kill count).

            scores.append(False, self.get_level(), self.get_score(),
self.get_kills())

            # Plays the menu music upon returning to menu.

            audio_cfg.play_music(Path.MENU_MUSIC_PATH)
            self.run = False

    # Stores information about the movement method selected by the user (i.e.
Mouse or Keyboard).

    def move(self):
        if(self.mouse_movement):
            self.move_with_mouse()
        else:
            self.move_with_keyboard()

    # Responsible for propaganting the player's laser beam through the screen.
    # Inputs include velocity and special parameters to distinguish the reward
for laser hitting different spaceship.
    def move_lasers(self, vel, objs):
        self.coolDown()
        for laser in self.lasers:
            laser.move(vel)

            # Removes the laser when they go off screen.

            if laser.off_screen(config.HEIGHT):
                self.lasers.remove(laser)
            else:
                for obj in objs:
                    if laser.collision(obj):

                        # Special reward if boss spaceship is defeated. Adds
1000 score points flat and increases kills by 1.
                        # If it is just a normal spaceship, 50 score points
and increases kills 1.
```

```
                        if obj.ship_type == 'boss':
                            if self.boss_max_health - 10 <= 0:
                                self.SCORE += 1000
                                self.KILLS += 1
                                objs.remove(obj)
                                self.boss_max_health = 100
                            else:
                                self.boss_max_health -= 10
                        else:
                            self.SCORE += 50
                            self.KILLS += 1

                            # Triggers enemy ship death explosion if laser
hits the enemy ship.

                            explosion = Explosion(obj.x, obj.y)
                            explosion_group.add(explosion)
                            objs.remove(obj)

                        if laser in self.lasers:
                            self.lasers.remove(laser)

    # Draws the health bar for the player's spaceship.

    def draw(self):
        super().draw()
        self.healthBar()

    # Defines the behaviour of the health bar for the player's spaceship.
    # It is thing and rectangular in shape. When full, it is green in color.
    # When health is lost, those parts are replaced with red.
    # When health bar becomes empty, a life is deducted and the bar is
replenished.

    def healthBar(self):
        x_offset, y_offset = self.ship_img.get_size()
        pygame.draw.rect(config.CANVAS, Colors.RED, (config.starting_x +
self.x -
                         x_offset/2, self.y + y_offset/2 + 10,
int(self.ship_img.get_width()), 10))
        pygame.draw.rect(config.CANVAS, Colors.GREEN, (config.starting_x +
self.x - x_offset/2, self.y +
                         y_offset/2 + 10, int(self.ship_img.get_width() *
(self.health/self.max_health)), 10))

# Defines the different types of enemy ships available.
```

```
# easy = Lanius Outrider
# medium = Zoltan Interceptor
# hard = Slug Instigator
# boss = Mantis Battlecruiser

class Enemy(Ship):
    TYPE_MODE = {
        'easy': (Image.EASY_SPACE_SHIP, Image.RED_LASER, 10),
        'medium': (Image.MEDIUM_SPACE_SHIP, Image.BLUE_LASER, 18),
        'hard': (Image.HARD_SPACE_SHIP, Image.GREEN_LASER, 25),
        'boss': (Image.BOSS_SHIP, Image.FLAME_LASER, 100)
    }

    ship_type = ''

    # Initializes the same functions as the player's ship had (such as moving
the ship, firing the lasers, moving the laser).

    def __init__(self, x, y, ship_type, health=100):
        super().__init__(x, y, health)
        self.ship_type = ship_type
        self.ship_img, self.laser_img, self.damage =
self.TYPE_MODE[self.ship_type]
        self.mask = pygame.mask.from_surface(self.ship_img)

    # Moves the ship with velocity defined in 'vel'.

    def move(self, vel):
        self.y += vel

    # Propagates enemy lasers through the screen with velocity specified in
'vel'.
    # Removes the lasers when they go off-screen.
    # If it hits player, display collision, explosion and reduce the health.

    def move_lasers(self, vel, obj):
        self.coolDown()
        for laser in self.lasers:
            laser.move(vel)
            if laser.off_screen(config.HEIGHT):
                self.lasers.remove(laser)
            elif laser.collision(obj):

                # Display collision if enemy laser hits the player.

                sm_explosion = Explosion(laser.x, laser.y, size=30)
                explosion_group.add(sm_explosion)
                obj.health -= self.damage
```

```
                self.lasers.remove(laser)

    # Fires the laser. Plays the laser firing audio audio and sets cooldown
timer to 1 after firing.
    def shoot(self):
        if self.cool_down_counter == 0 and self.y > 0:
            Sound.ENEMY_LASER_SOUND.play()
            laser = Laser(self.x, self.y, self.laser_img)
            self.lasers.append(laser)
            self.cool_down_counter = 1
```

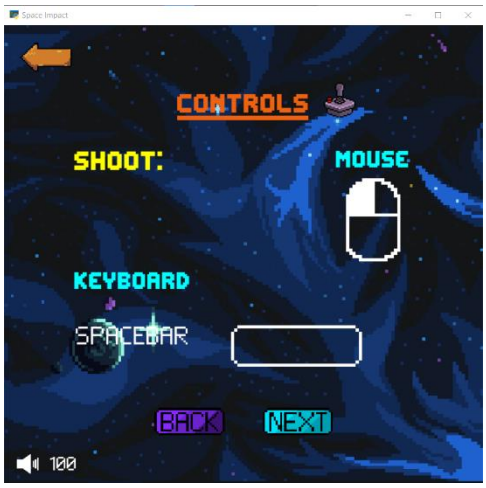
```
# Filename: ./models/slider.py

# Function: Model file that implements the slider system found on the settings
page.

# Importing the required code from other modules of the game.

from config import config
from constants import Colors

# Importing all modules required for proper functioning of the code in this
file.

import pygame

# Initializes all imported pygame modules in a convenient way to get
everything started.

pygame.init()

# Loads & initializes font to be used (Verdana of size 12 in this case).

font = pygame.font.SysFont("Verdana", 12)


class Slider():

    # Initializes the slider component. Inputs include start value, position
on screen, max. or min. at slider positions left/right.
    def __init__(self, name, val, maxi, mini, pos):
        self.val = val   # Starting value.
        self.maxi = maxi # Maximum value is at slider position right.
        self.mini = mini # Minimum value is at slider position left.
        self.xpos = pos  # At x-coordinate on screen.
        self.ypos = 250  # Specifies the y-coordinate.
        self.surf = pygame.surface.Surface((200, 50))
        self.hit = False  # Hit attribute indicates slider movement due to
mouse interaction.

        # Static graphics, which is essentially the slider background.
        pygame.draw.rect(self.surf, Colors.WHITE, [10, 30, 160, 5], 0)

        # Dynamic graphics, which is essentially the button surface.
        self.button_surf = pygame.surface.Surface((20, 20))
        self.button_surf.fill(Colors.TRANS)
        self.button_surf.set_colorkey(Colors.TRANS)
        pygame.draw.circle(self.button_surf, Colors.BLACK, (10, 10), 6, 0)
        pygame.draw.circle(self.button_surf, Colors.WHITE, (10, 10), 4, 0)
```

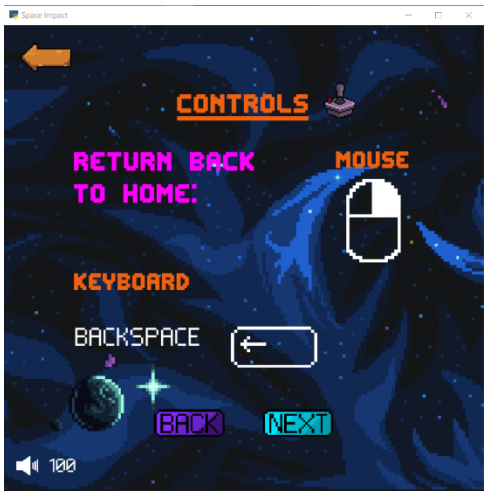
```
    def draw(self):

        # Combination of static and dynamic graphics in a copy of the basic
slide surface.

        # Static
        surf = self.surf.copy()

        # Dynamic
        pos = (10+int((self.val-self.mini)/(self.maxi-self.mini)*100), 33)
        self.button_rect = self.button_surf.get_rect(center=pos)
        surf.blit(self.button_surf, self.button_rect)

        # Moving button box to correct screen position.
        self.button_rect.move_ip(self.xpos, self.ypos)

        # Screen
        config.CANVAS.blit(surf, (self.xpos, self.ypos))

    def move(self):

    # The dynamic part; reacts to movement of the slider button.

        self.val = (pygame.mouse.get_pos()[
                    0] - self.xpos - 10) / 80 * (self.maxi - self.mini) +
self.mini
        if self.val < self.mini:
            self.val = self.mini
        if self.val > self.maxi:
            self.val = self.maxi
```

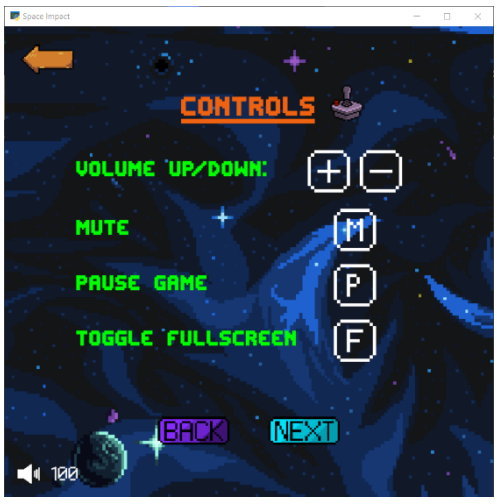
4.0 SCREENSHOTS

Main Menu

Settings Screen

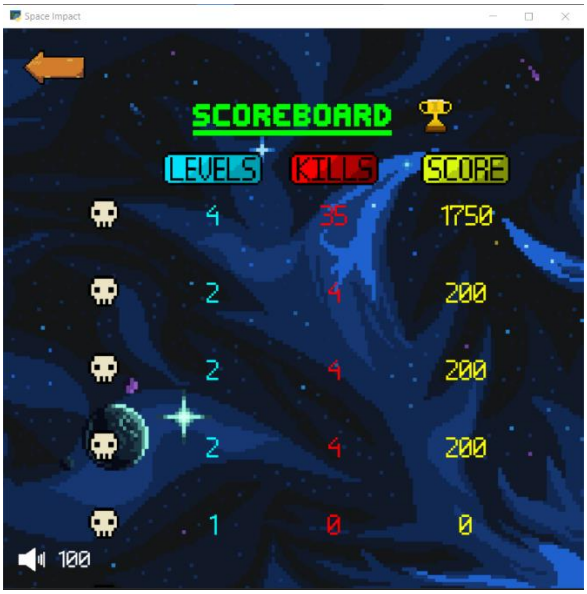






Control info screen(s)

Ships screen



Scoreboard screen

Gameplay screen

Pause screen

5.0 CONCLUSION

We were able to implement a Multimedia game project using various concepts we've learnt as part of our course while also being able to improve upon our subject knowledge. We had a fun time working on the project and we're confident on how to use pygame to implement powerful Multimedia applications such as this. We'd like to thank Priyambada ma'am, our course faculty for giving us this wonderful opportunity.

END OF PROJECT REPORT