“SPACE SHOOTER”
Using Python

A MINI PROJECT REPORT

Submitted by:

ANCHAL SINHA
RAJAT
DEBASHISH HEMBRAM

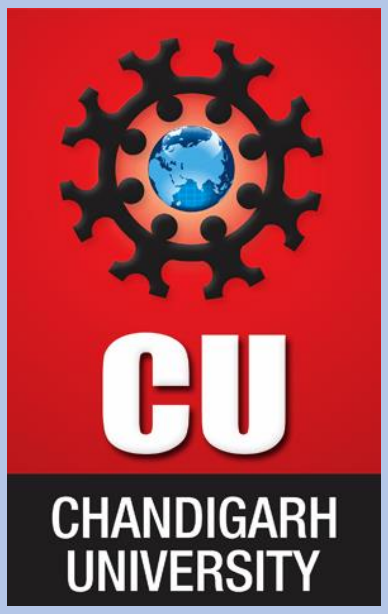
in our partial fulfillment for the award of the degree

of

B.SC. CS

in

BRANCH OF STUDY



UNIVERSITY INSTITUTE OF COMPUTING

CHANDIGARH UNIVERSITY

November, 2021

CERTIFICATE

Certified that this project report "SPACE SHOOTER" is the bona fide work of "Debashish Hembram, Anchal Sinha and Rajat" who carried out the project work under my supervision.

Ms. Saranjeet Kaur
(Guide)

Ms. Manisha Malhotra
(UIC, Head of Department)

ACKNOWLEDGEMENT

We would like to express our exceptional thanks of gratitude to our project guide Ms. Saranjeet Kaur as well as our project coordinator Ms. Pavandeep Kaur who gave us the golden opportunity to do this wonderful project on the topic “Space Shooter”, which also helped us in doing a lot of research and all three of us came to know about so many new things. We are really thankful to them.

Then, we would also like to thank our friends and family who helped us in finalizing this project within the limited time period.

Date: 5th November, 2021

NAME:
ANCHAL SINHA
DEBASHISH HEMBRAM
RAJAT

UIDs:
19BSC1013
19BSC1030
19BSC1026

TABLE OF INDEX Page No.

ABSTRACT

In our project, we have a player spaceship which is lost deep inside the space where it is being obstructed by the meteorites and its sole objective is to survive the incoming waves of meteors as long as possible. The player can move in only one axis of freedom. There are four types of meteors, big, medium, small and tiny. The meteors will have basic Artificial Intelligent for the judgement of the player's movement and unique behavior for each meteor.

The main objective of this project is to build a game from scratch in Python and Pygame module. The principle of the game, is based upon Pygame which is a Python module built for game development. The end product is a game, where the goal of the player is to destroy all the incoming meteors, picking up buffs or debuffs along the way. The report will provide an explanation of the technicality of the project, as well as the reasons this game was designed & built in the first place. The report starts by creating a context for this project and will continue guiding the reader with the complete implementation.

PREFACE

This project comprises of a lost space ship in the space finding its way to home, and it's objective is to survive the incoming waves of meteorites. Various graphics such as the player ship, explosions, lasers, meteors etc. are being used in the game. We chose this project as all of us like to play video games and wanted to design a mini game of our own, and in doing so we learned various things necessary for creating video games like the coding part, things to consider while making the interface, user requirements and much more. The reader would get to know the things required to create a game of their own without much research, as the report is written in very simple and easily understandable terms.

Ch 1. INTRODUCTION

The game “Space Shooter/Invader” is an arcade video game, which came into an existence in 1978 designed by Tomohiro Nishikado, manufactured and sold by Taito Japan, and licensed by the Midway of bally for overseas distribution. Within all the shooter genre, Space Shooter was the first fixed shooter. The aim is to defeat waves of descending meteors with lasers moving in all the possible direction and to simultaneously earn as many points as possible.

➢ ABOUT SYSTEM / BACKGROUND:

The project file contains python scripts (space_shooter.py). Explaining about the gameplay, the player has to shoot the meteor using spacebar button. The player must align with the meteor and select the spacebar to destroy the meteor. The player has to shoot and destroy all the incoming meteor to win the game. The controls are very simple, as you can use your left and right arrow keys for the player ship movement and spacebar for shooting. The gameplay design is very simple that the user won’t find it difficult to use and understand.

➢ OBJECTIVES:

The aim is to defeat waves of descending meteors with lasers moving in all the possible direction and to simultaneously earn as many points as possible. The objective of the game is simply to destroy a formation of advancing meteorites. The further the game proceeds, the obstructions get tougher and tougher with meteors approaching at faster pace more frequently. The number of points obtained per enemy destroyed increases. The player should survive as long as possible to maximize his or her score.

➢ APPLICABILITY:

The game allows the player to play the game, pickup buffs to power-up and heal. The player controls character movements over obstacles and defeat enemies. Player character will lose a life or reduces its shields when collided with enemies or lethal obstacle.

➢ PURPOSE:

The purpose of this game is to attach the human emotions with this video game. Although the game, we created is not that much advanced but it will definitely become a reason for the stress relief.

➢ SCOPE:

This document describes the design of space shooter. It contains the requirements, theory of operation and dependencies. Schematic diagrams, and code are used to demonstrate the architecture described. A list of protocol used to test functionality of the Space Shooters are explained. The mechanical and hardware description of the Space Shooter is not included.

➢ ORGANISATION OF REPORT:

The Report has been made and organised as per the given guidelines in the file "Format of Synopsis and Report (Project).docx", with a little add-on of our own.

➢ GAME FEATURES:

- Scoreboard to show how much scores you've obtained so far.
- Power ups like:
    - Shied: It increases the spaceship's life.
    - Bolt: It increases the shooting capability of the spaceship by firing 2-3 bullets instead of one at time.
- Audio and sounds which are customized and sprite animation things like:
    - Meteorite explosions.
    - Bullets shooting.
    - Player explosion.
    - Custom sounds before and after the game. (eg: get ready to play)
- 3 lives are allotted per game.
- After the game ends, it brings you back to the main menu.

➢ CONTROLS:

Instructions	**Buttons**
Move left	Left
Move right	Right
Fire bullets	Spacebar
Quit Game	Q

Ch 2. SYSTEM ANALYSIS

➢ IDENTIFICATION OF NEED:

As this game is a classic retro game which is having slight difference. According to our research in every relevant stuff regrading space shooter, they are approaching different ideas. For example, some are including meteorites, pop-up dialog boxes before and after the game, customized sounds, directions for moving etc. and some are not. So, we have planned to do a bit change by adding more effects, directions, sounds, stuffs which will level-up our game.

The image of the game from where we took the idea is given below:

Figure: Old version of space shooter

This is the old version of this space shooter where the spaceship has shields above it, and it's using very old graphics elements in it.

➢ PRELIMINARY INVESTIGATION:

In our game,

- We have added the customized sound effects.
- We are using new graphic models in the game.
- New types of incoming obstructions like meteors of various sizes.

➢ FEASIBILITY STUDY:

As we already know by now, this project comprises of a lost space ship in the space finding its way to home, and it's objective is to survive the incoming waves of meteorites. Various graphics such as the player ship, explosions, lasers, meteors etc. are being used in the game. We chose this

project as all of us like to play video games and wanted to desing a mini game of our own, and in doing so we learned various things necessary for creating video games like the coding part, things to consider while making the interface, user requirements and much more.

Rendering based on level of detail: it is possible to achieve greater performance with more complex models if we control the level of detail in models being rendered depending on where they are located in world space. For example, we could have used models with different numbers of vertices. When the models are not visible to the player, the program would render the model with the least number of vertices and it gets closer to the player. We could gradually use models with higher number of vertices as this would potentially improve the experience of the player while making sure that performance is not compromised at all.

Since this project is being made for experience and to gain knowledge and we found all the necessary resources and technologies required for the game, therefore there was no investment made in the form of capital and we don't expect any kind of return for the project.

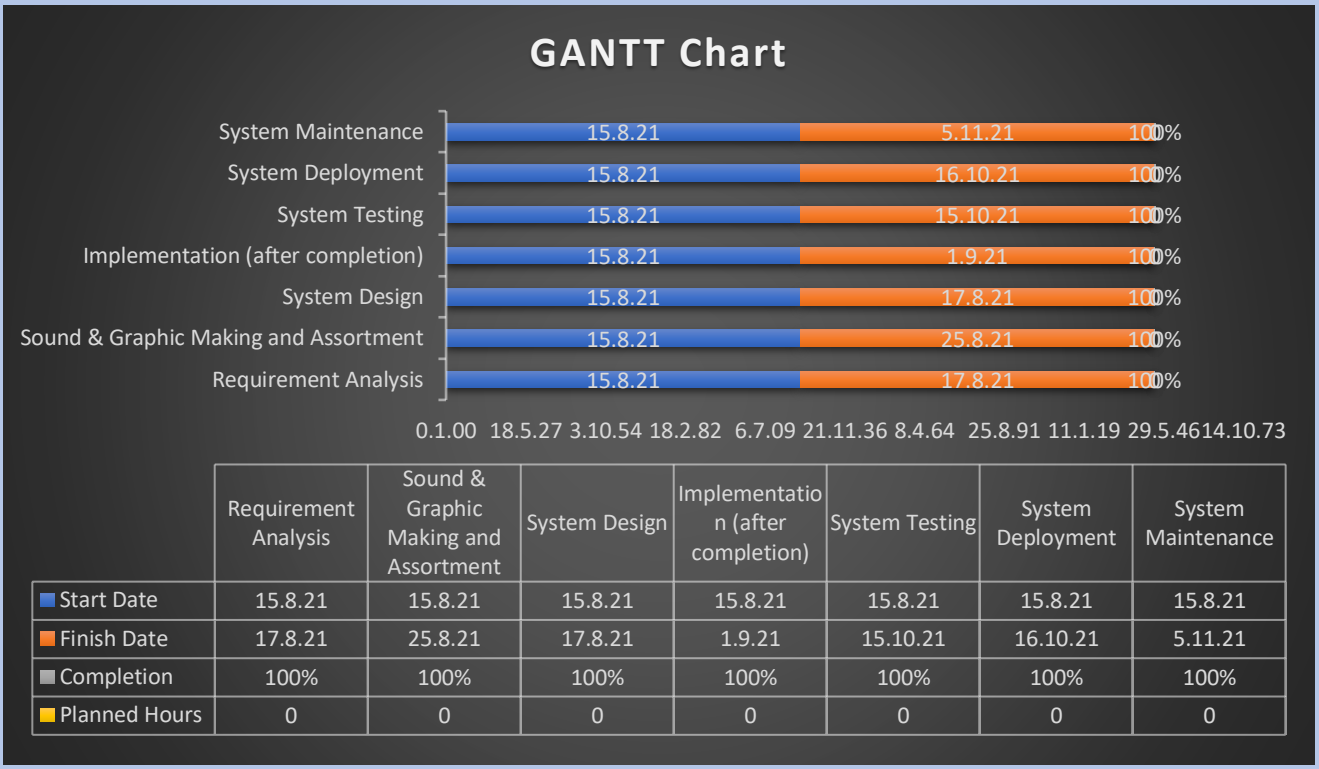
➢ PROJECT PLANNING:

By building this space shooter game, we'll get to know the working and planning of this game accompanied by the usage of backends like coding, and loading will be imparted through this game. This will make more convenient using this report to follow the steps and procedures written and it would definitely make the user aware of the Python and Pygame.

Each project group (a group of three) must submit the deliverables and comply with the requirements in Section 7. In addition to the deliverables needed to submit in the earlier submissions, the final submission must include:

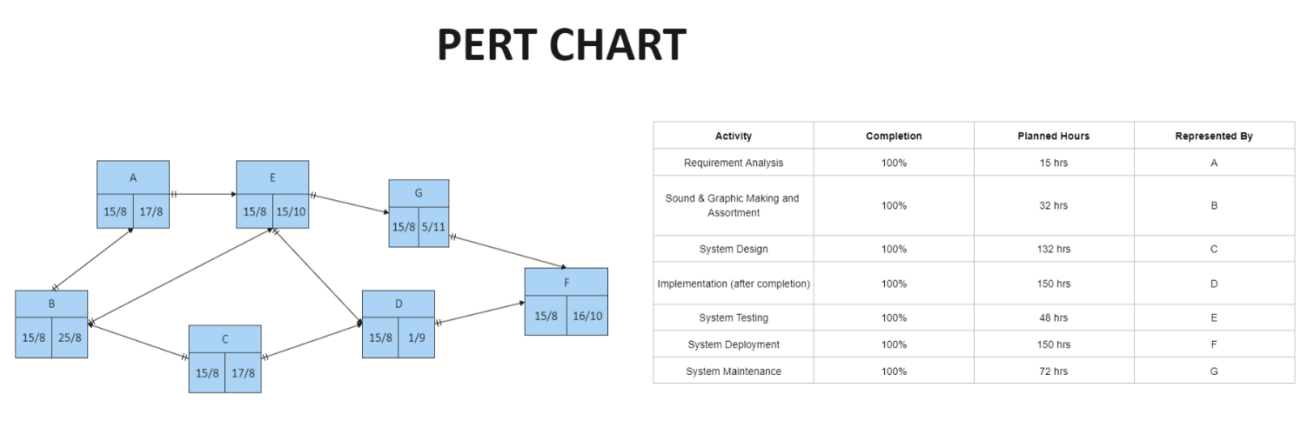
- A Project Report presenting the problem, its analysis and specification; the conceptual model of the domain; an overview of how the code is structured; the 5 various classes/class hierarchies and their responsibilities; the dynamic behavior of key parts of the system; and a critique of both the final functionality achieved and the object-oriented design.
- A form should be signed by all project partners stating which person did which parts of the project and specifying each person's discretionary mark which will be combined with project marks. If no discretionary marks are specified, or if the form is not signed by all the group members, then the marks will be forfeited.
- A technical reference manual explaining the source code for other programmers who might wish to understand and modify it.

➢ PROJECT SCHEDULING:



	Requirement Analysis	Sound & Graphic Making and Assortment	System Design	Implementation (after completion)	System Testing	System Deployment	System Maintenance
Start Date	15.8.21	15.8.21	15.8.21	15.8.21	15.8.21	15.8.21	15.8.21
Finish Date	17.8.21	25.8.21	17.8.21	1.9.21	15.10.21	16.10.21	5.11.21
Completion	100%	100%	100%	100%	100%	100%	100%
Planned Hours	0	0	0	0	0	0	0

Fig. GANTT Chart



Activity	Completion	Planned Hours	Represented By
Requirement Analysis	100%	15 hrs	A
Sound & Graphic Making and Assortment	100%	32 hrs	B
System Design	100%	132 hrs	C
Implementation (after completion)	100%	150 hrs	D
System Testing	100%	48 hrs	E
System Deployment	100%	150 hrs	F
System Maintenance	100%	72 hrs	G

Fig. PERT Chart

➢ SOFTWARE REQUIREMENT SPECIFICATIONS (SRS):

In this game, we've used the platforms like, IDLE, Visual Studio Code and Sublime text editor. There were several optimization and smoothing techniques that were used to ensure that the game play was possible efficiently.

About IDLE:

IDLE is a file editor for Python which is used for software and application development. To know more, click the link given: [cpython/Lib/idlelib at 3.9 · python/cpython · GitHub]

Remark about opening .py files directly:
It is not recommended to open python files ending with .py just by double clicking on them. Instead, open the programs using the IDLE. This is because the behavior you get can be unexpected, although it won't do any harm.
Start → All Programs → Python 3.9 → IDLE (Python GUI)

STEPS ARE GIVEN FOR INSTALLING IDLE:

STEP 1: Click the link ([IDLE — Python 3.9.7 documentation]) to download.
STEP 2: When the file will be downloaded kindly follow the instructions which will be navigated by its own.
STEP 3: The below diagram is the latest version of IDLE.

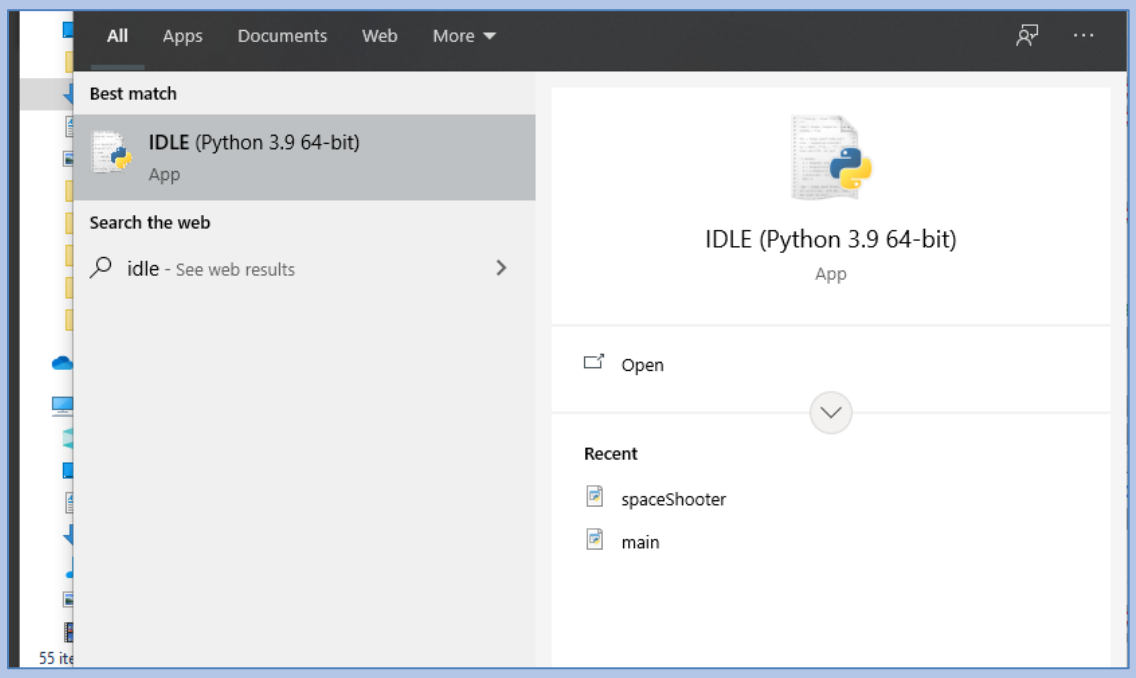
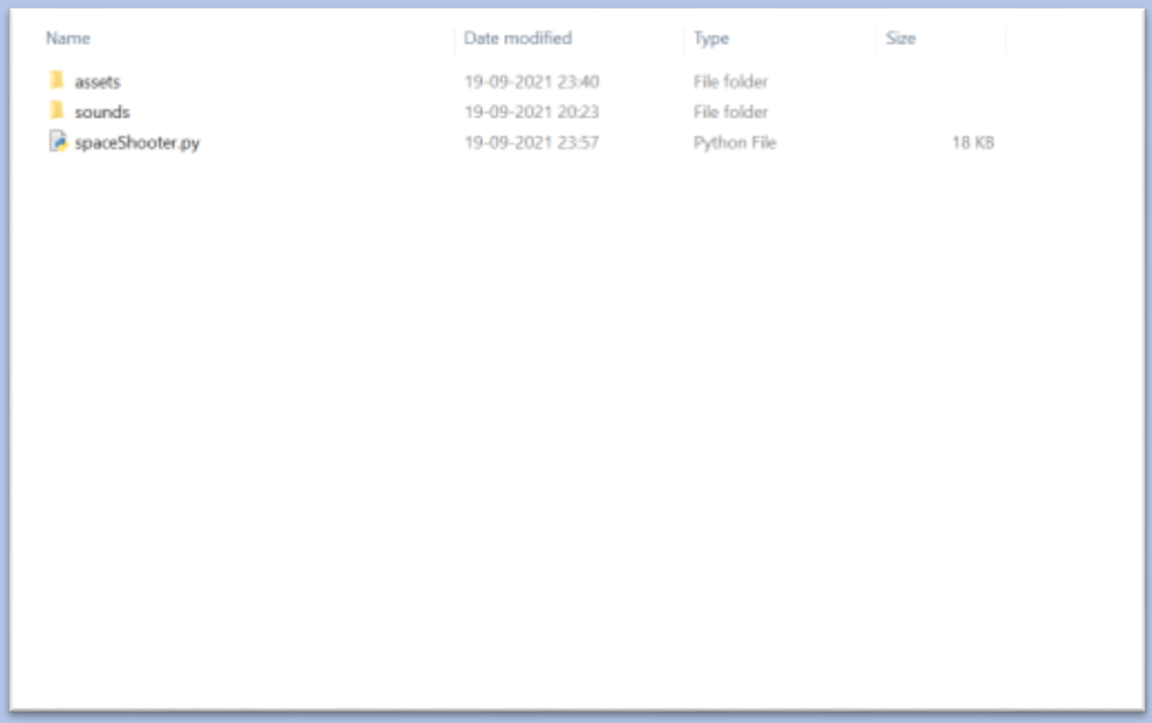


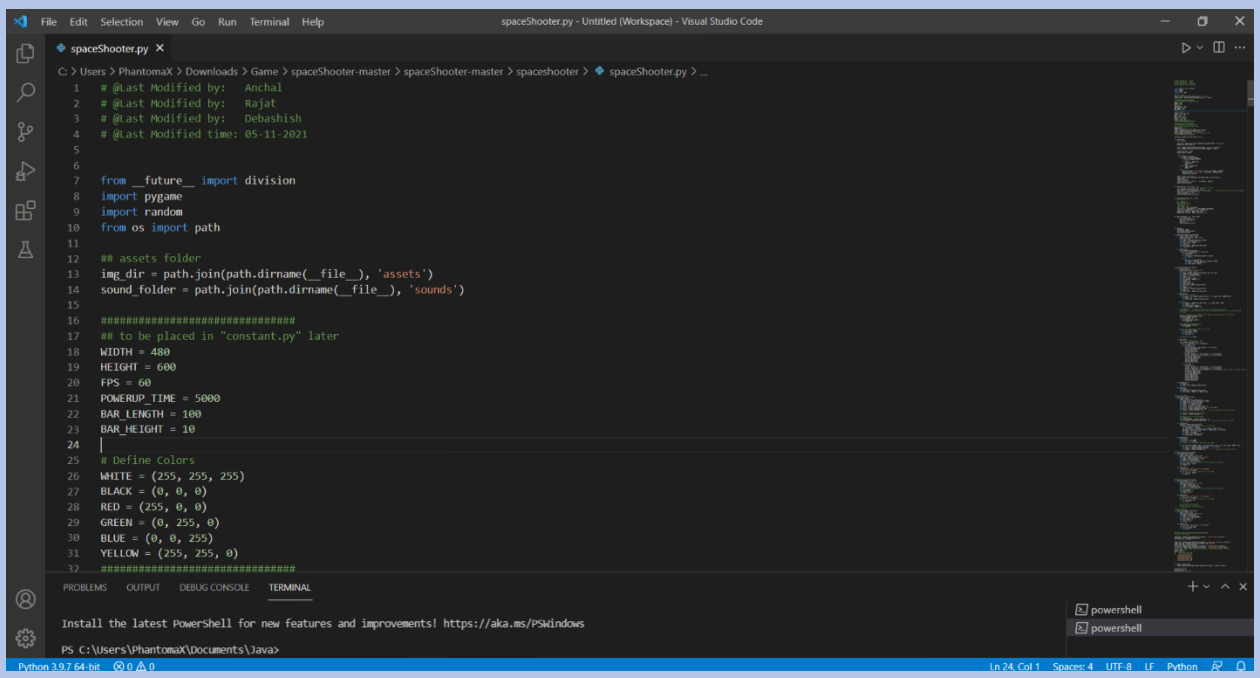
Figure: Opening IDLE (after installation)



These are the assets, files and tools we've used throughout the program.

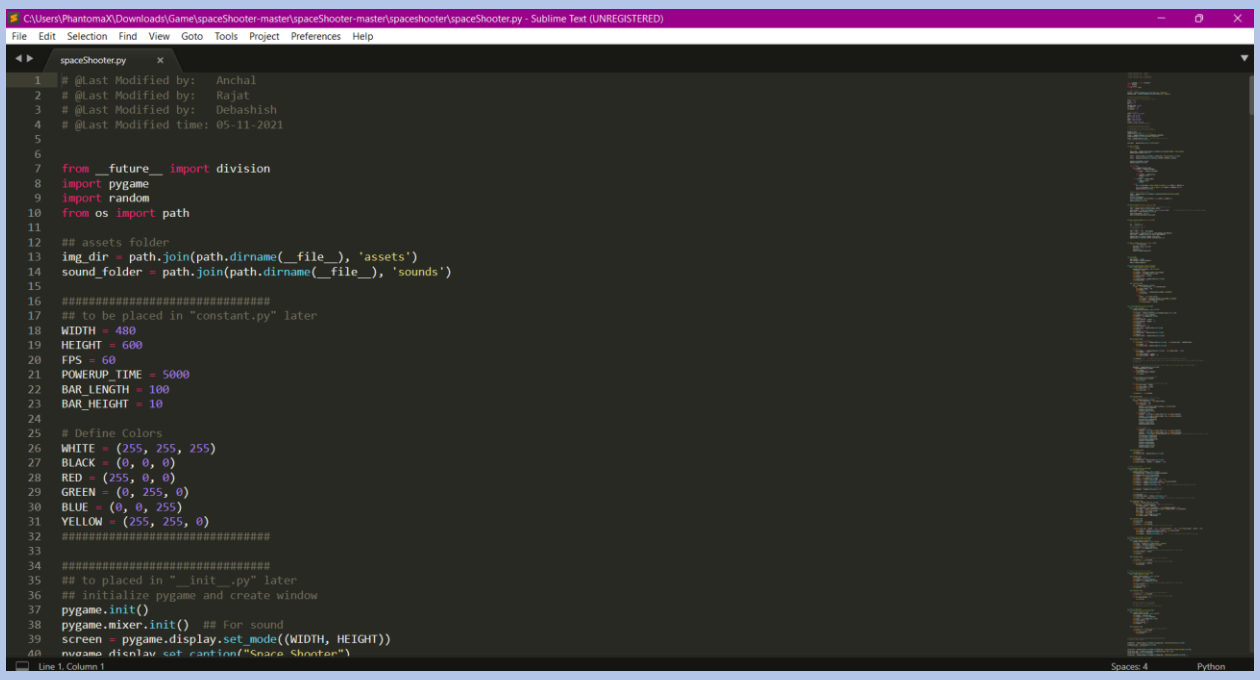
About Microsoft Visual Code:

Microsoft Visual Code is generally a text editor which can be used as any of the programming language's running environment after installing the pre requisite development kits for the same. To know more, click the link below: https://code.visualstudio.com/



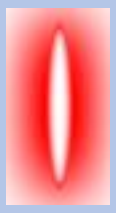
About Sublime Text Editor:

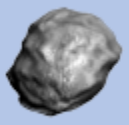
Sublime Text Editor is also a text editor which is also used for coding, with various features of it's own. To know more, click the given link: https://www.sublimetext.com/



➢ SOFTWARE ENGINEERING PARADIGM APPLIED:

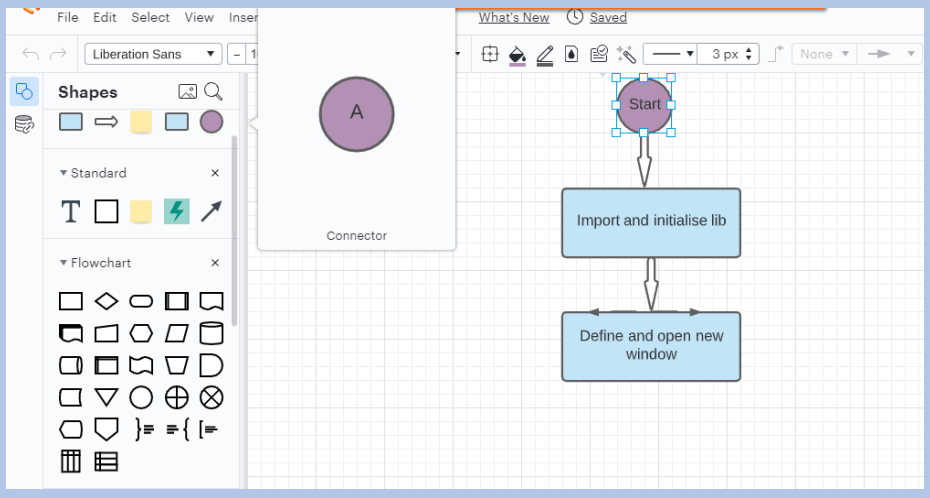
The below paradigms have been applied through this game. Further specifications are given with the coding in the upcoming module.

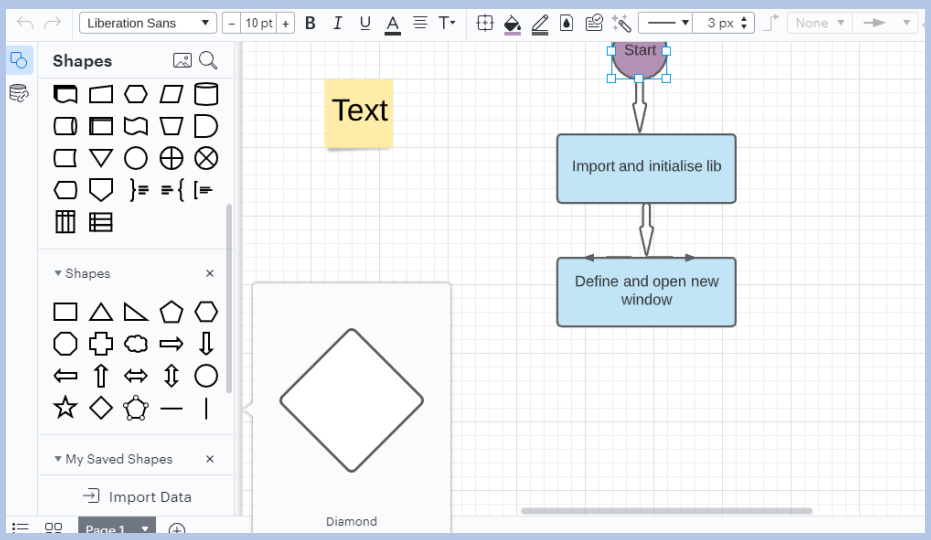


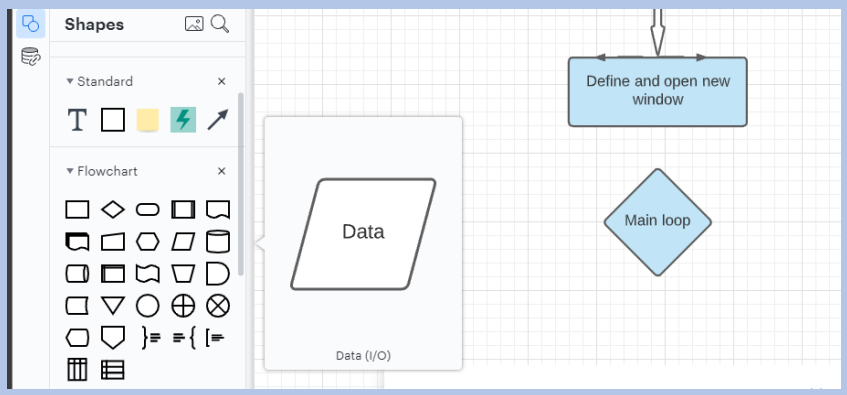


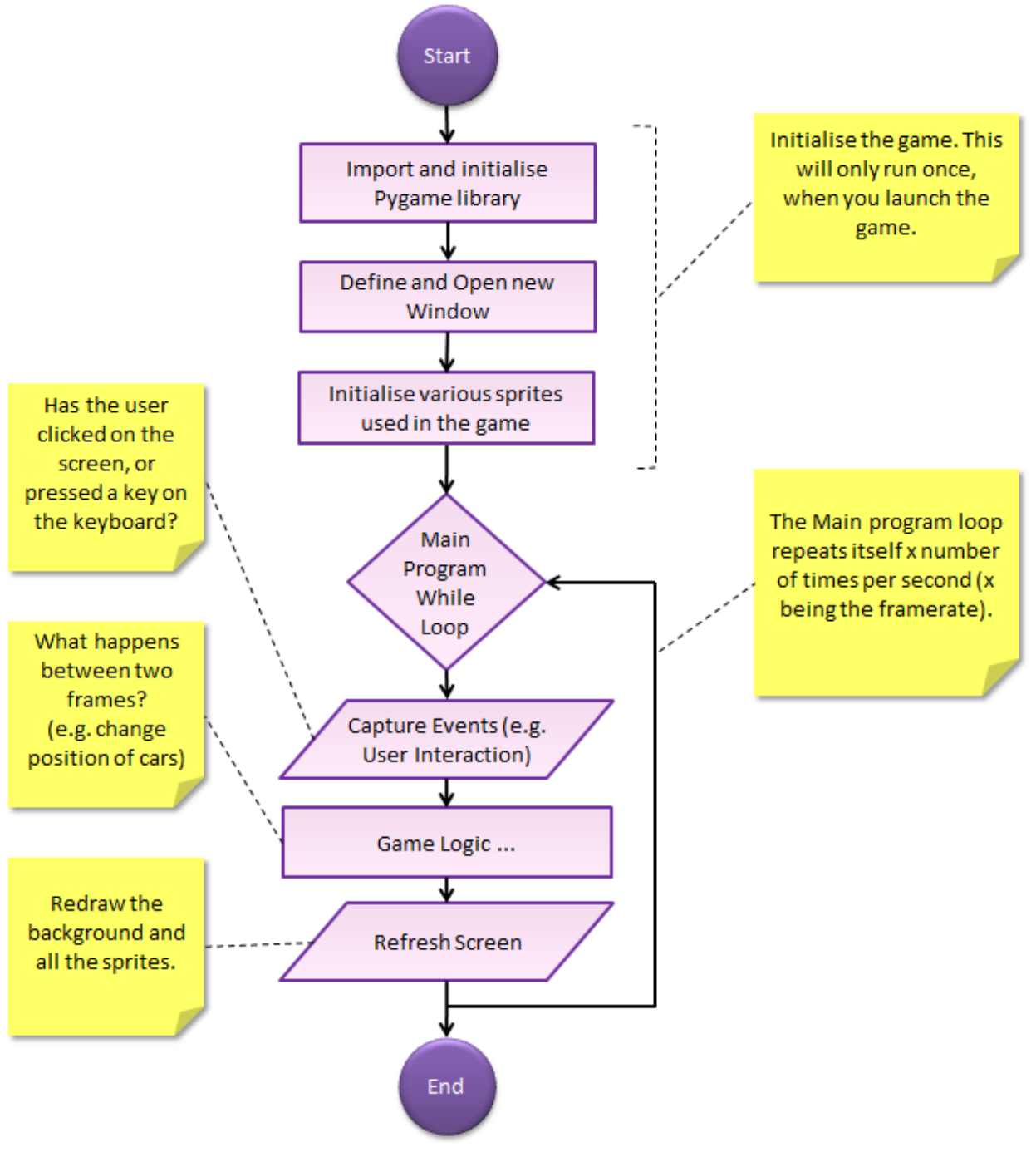
➢ DATA MODELS:

We've used process, connectors, data, line etc. using **https://lucid.app/lucidchart/**








Start
Import and initialise Pygame library
Define and Open new Window
Initialise various sprites used in the game
Main Program While Loop
Capture Events (e.g. User Interaction)
Game Logic ...
Refresh Screen
End
Initialise the game. This will only run once, when you launch the game.
Has the user clicked on the screen, or pressed a key on the keyboard?
What happens between two frames? (e.g. change position of cars)
Redraw the background and all the sprites.
The Main program loop repeats itself x number of times per second (x being the framerate).

Ch 3. SYSTEM DESIGN

We will now discuss the features, the main body of the project where we will be acquainted by all the features, designs of the game.

➢ MODULARISATION DETAILS:

We are using the Agile methodologies as they are very popular method of developing high quality software. They are both incremental & iterative in nature. Iterative meaning that the code has been passed through the phases of the software development lifecycle take place, multiple times. After completion of each iteration a working build is produced that has the subset of the total functionality that is required. Each iteration helps in an increment in the functionality of the code and iterations continue until the final required software is achieved. In order to complete this style of development, there will be two intervening time submissions and a final submission. The first two submissions are the submissions of the work-in-progress of the project. The final submission will require the submission of the completion of the project with its deliverables.

➢ DATA INTEGRITY AND CONSTRAINTS:

Though we haven't used any databases but in making of this project, we tried to maintain the data integrity and moral principles impeccably. Integrity like: started the project following the SDLC from development to testing, from coding to debugging, from designing to searching etc.

There were also some the minor constraints throughout the program, which we've managed to do so. Constraints like: there were many error we have faced during coding, plagiarism is also one of the factor which of course became the hinderance of the length of this report

1. Variable Frame Rate Rendering:

Since the performance of such a game is dependent on the system's capabilities (especially the graphics card), it is likely to see different frame rates on differently equipped systems. In order for the game dynamics to operate at a constant speed, the time step between each frame render is found. This change in time between frames is used to update the game objects to ensure close to constant movement rates at different rendering rates. On a faster system the time between frames will be small, and so will the movement between each frame; On a slower system, the time between frames will be higher, causing high changes (low performance/less smooth) from one frame to the next. It is possible that on a slower system that the time steps are so great that the game will be running in slow pace. Care must be made to ensure that the game runs at reasonable rates on most modern systems.

2. Vertex Arrays:

The game requires that lots of models be rendered on the screen simultaneously. If the rendering of these models were done on a vertex-by-vertex basis using a multiple number of OpenGL calls, the efficiency and performance of the game suffers heavily as we discovered. Vertex Arrays allow us to replace hundreds of OpenGL calls with only a very few calls that work on arrays of data related to the vertices.

3. Model Complexity:

With the original unnecessarily complex models (which are very computationally challenging for integrated graphics cards), it was shown that the frame rate was less than 10 frames/sec, which made the game unplayable. After the new models were introduced, the frame rate went up to over 60 FPS, even on a laptop's integrated graphics card.

➢ USER INTERFACE DESIGN:

AUDIO:

There are two types of audios used in the game. MP3 audio for music, and WAVE files for sound effects such as lasers and explosions. The WAVE files are found on FindSounds - Browse for sounds, an online audio search engine used for finding publicly available sounds.

USER INPUT:

One of the advantages of the game is the use of non-standard input methods. The mouse and keyboard input exist, they are used for debugging during development. The main inputs required for the gameplay are the controls for player movement, and for shooting. As per the given parameters of the game, the movement is just along a single dimension (x axis only), and a single button for shooting is required.

FIRE MECHANISM:

In most games, the trigger for firing is usually a key (such as spacebar) or a mouse click. At first, we decided that the lasers will be triggered with a specific phrase such as “fire!”, but such analysis of the audio would create difficulties that are way past the scope of the project. Therefore, it is possible for the lasers to be activated using any particular sound. However, as mentioned before, the ability to fire using a mouse buttons/keyboard buttons still exists if the player chooses to do so.

MENU SYSTEM:

A menu system is developed for the player to start the game by pressing ENTER or quit by pressing Q. There is only one menu screen available to the player presented at the beginning of the game. These menus are discussed below.

MAIN MENU:

Upon starting the game, the player is presented with the main menu where the player can select one of two options:

START GAME:

- The game can be played by pressing the ENTER button.
- In Fig. 3, you can see the options to begin and quit the game application.

```
title = pygame.image.load(path.join(img_dir, "main.png")).convert()
 title = pygame.transform.scale(title, (WIDTH, HEIGHT), screen)

 screen.blit(title, (0,0))
 pygame.display.update()

 while True:
     ev = pygame.event.poll()
     if ev.type == pygame.KEYDOWN:
         if ev.key == pygame.K_RETURN:
             break
         elif ev.key == pygame.K_ESCAPE:
             pygame.quit()
             quit()
     else:
         draw_text(screen, "Press [ENTER] To Begin", 30, WIDTH/2, HEIGHT/1.55)
         draw_text(screen, "or [ESC] To Quit", 30, WIDTH/2, (HEIGHT/1.55)+40)
         pygame.display.update()
```

Figure: This is the code for the same.

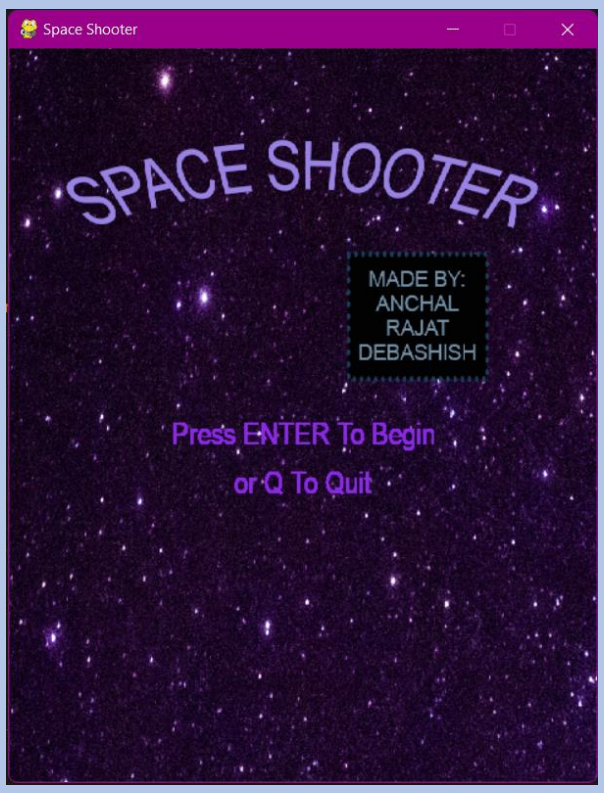


Figure: This is the window which appears after starting the game application.

QUIT GAME:

- Q button can be selected by the user to exit the game application.
- From the above image, we can also see the option for quitting.

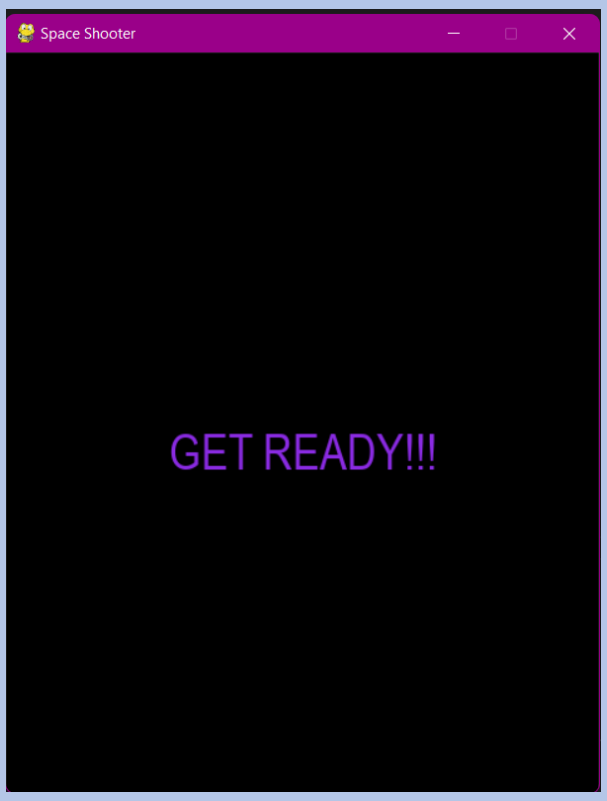


Figure: This is the GET READY notification while starting the game.

3.2 GAME OVER:

- This menu is displayed to the player when the game is over.
- The game will be over for the following reason: - Player has run out of lives (3 of them) The player's score is displayed on the screen and the menu options allow the player to either begin the game again or exit the game by pressing Q.

SHOOTING:

- Shooting the players laser cannons and missiles. These are the rules governing shots.
- If a player's shot hits a meteor both the meteor and the shot are destroyed.
- If a meteor hits a player's laser cannon, then both the laser cannon and the missile are destroyed.
- If a player's shot hits a meteor, both are destroyed.

➢ DATABASE DESIGN, PROCEDURAL DESIGN:

Assets, deliverables and software we used could be explained in this section.
This section shows in detail the technical aspects of the game's implementation.

GAME OBJECTS:

The game objects in the game refer to the major moving objects in the game. These include the player ship, missile, meteors, lasers, explosion hits, main page and power-ups.

PLAYER SHIP:

MISSILES:

LASERS:

EXPOSIONS:

Regular Explosions:

Figure: Regular explosion 1

Figure: Regular explosion 2

Figure: Regular explosion 3

Figure: Regular explosion 4

Figure: Regular explosion 5

Figure: Regular explosion 6

Figure: Regular explosion 7

Figure: Regular explosion 8

Figure: Regular explosion 9

Sonic Explosions:

Figure: Sonic explosion 1

Figure: Sonic explosion 2

Figure: Sonic explosion 3

Figure: Sonic explosion 4

Figure: Sonic explosion 5

Figure: Sonic explosion 6

Figure: Sonic explosion 7

Figure: Sonic explosion 8

Figure: Sonic explosion 9

METEORS:

Figure: Small meteor

Figure: Small meteor

Figure: Medium meteor

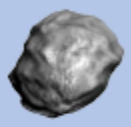


Figure: Medium meteor

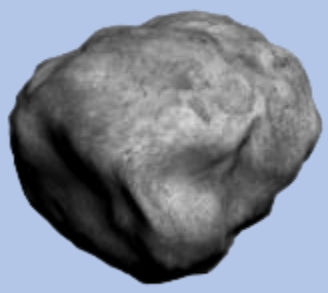


Figure: Big meteor

Figure: Big meteor

Figure: Tiny meteor

MAIN PAGE:

Figure: Home Page

POWERUP:

Figure: Shield

Figure: Power Boost

External Libraries:

We have used many external libraries for developing our version of Space Invaders. Following is the list and a brief explanation of the various libraries used in the project.

SDL_AudioIn:

Provides audio input support for SDL. Used to capture samples from the microphone to implement sound triggered shooting. Below is the link from where we've used the sounds.

https://www.findsounds.com/

```
405  ### Load all game sounds
406  shooting_sound = pygame.mixer.Sound(path.join(sound_folder, 'pew.wav'))
407  missile_sound = pygame.mixer.Sound(path.join(sound_folder, 'rocket.ogg'))
408  expl_sounds = []
409  for sound in ['expl3.wav', 'expl6.wav']:
410      expl_sounds.append(pygame.mixer.Sound(path.join(sound_folder, sound)))
411  ## main background music
412  #pygame.mixer.music.load(path.join(sound_folder, 'tgfcoder-FrozenJam-SeamlessLoop.ogg'))
413  pygame.mixer.music.set_volume(0.2)      ## simmered the sound down a little
414
415  player_die_sound = pygame.mixer.Sound(path.join(sound_folder, 'rumble1.ogg'))
416  ################################################
```

Figure: Sounds used in the game.

SDL_Image:

This library is used to load files of different types to the game for texturing purposes. This saves us the time to write various custom functions that would do this. This is the site from where we've used the graphic elements and altered them using Photoshop:

https://opengameart.org/

```
347  ################################################
348  ## Load all game images
349
350  background = pygame.image.load(path.join(img_dir, 'starfield.jpg')).convert()
351  background_rect = background.get_rect()
352  ## ^^ draw this rect first
353
354  player_img = pygame.image.load(path.join(img_dir, 'playerShip1_orange.png')).convert()
355  player_mini_img = pygame.transform.scale(player_img, (40, 23))
356  player_mini_img.set_colorkey(BLACK)
357  bullet_img = pygame.image.load(path.join(img_dir, 'laserRed16.png')).convert()
358  missile_img = pygame.image.load(path.join(img_dir, 'missile.png')).convert_alpha()
359  # meteor_img = pygame.image.load(path.join(img_dir, 'meteorBrown_med1.png')).convert()
360  meteor_images = []
361  meteor_list = [
362
363      'meteorBrown_med1.png',
364      'meteorBrown_med3.png',
365      'meteorBrown_small1.png',
366      'meteorBrown_small2.png',
367      'meteorBrown_tiny1.png'
368  ]
```

Figure: Images used in the game.

Lib3DS:

This library is used to load the different types of 3D models that our game requires. We selected the 3DS file format for the models used in our game as there are lots of resources available on the internet to handle 3DS files. The Lib3DS library helps with the rendering of the various meshes and error handling.

```
6
7   from __future__ import division
8   import pygame
9   import random
10  from os import path
```

Figure: Libraries used in this game

```
43
44  font_name = pygame.font.match_font('Arial')
```

Figure: Library for font in python

Above shown library allows us to render fonts on our screens. It provides sufficient features for us to render fonts for our game menus as well as for showing the status of the player on screen. There were several optimization and smoothing techniques that were used to ensure that the game play was possible efficiently. They are discussed below.

➢ TEST CASES:

Test case can't be specified in this small piece of report. Though we've provided the samples of coding but it can't be specified in this section solely as it is containing more than 500 lines of coding and lots of test cases.

Ch 4. CODING

➢ Complete Project Coding:

```
# Last Modified by:   Anchal, Debashish, Rajat
# Last Modified time: 05-11-2021

from __future__ import division
import pygame
import random
from os import path

## assets folder
img_dir = path.join(path.dirname(__file__), 'assets')
sound_folder = path.join(path.dirname(__file__), 'sounds')

##############################
## to be placed in "constant.py" later
WIDTH = 480
HEIGHT = 600
FPS = 60
POWERUP_TIME = 5000
BAR_LENGTH = 100
BAR_HEIGHT = 10

# Define Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
YELLOW = (255, 255, 0)
##############################

##############################
```

```
## to placed in "__init__.py" later
## initialize pygame and create window
pygame.init()
pygame.mixer.init()  ## For sound
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Space Shooter")
clock = pygame.time.Clock()     ## For syncing the FPS
###############################

font_name = pygame.font.match_font('Arial')

def main_menu():
    global screen

    menu_song = pygame.mixer.music.load(path.join(sound_folder, "menu.ogg"))
    pygame.mixer.music.play(-1)

    title = pygame.image.load(path.join(img_dir, "main.png")).convert()
    title = pygame.transform.scale(title, (WIDTH, HEIGHT), screen)

    screen.blit(title, (0,0))
    pygame.display.update()

    while True:
        ev = pygame.event.poll()
        if ev.type == pygame.KEYDOWN:
            if ev.key == pygame.K_RETURN:
                break
            elif ev.key == pygame.K_q:
                pygame.quit()
                quit()
```

```
        elif ev.type == pygame.QUIT:
                pygame.quit()
                quit()
        else:
            draw_text(screen, "Press ENTER To Begin", 25, WIDTH/2, HEIGHT/2)
            draw_text(screen, "or Q To Quit", 25, WIDTH/2, (HEIGHT/2)+40)
            pygame.display.update()

    #pygame.mixer.music.stop()
    ready = pygame.mixer.Sound(path.join(sound_folder,'getready.ogg'))
    ready.play()
    screen.fill(BLACK)
    draw_text(screen, "GET READY!!!", 40, WIDTH/2, HEIGHT/2)
    pygame.display.update()


def draw_text(surf, text, size, x, y):
    ## selecting a cross platform font to display the score
    font = pygame.font.Font(font_name, size)
    text_surface = font.render(text, True, (138,43,226))       ## True denotes the font to be anti-aliased
    text_rect = text_surface.get_rect()
    text_rect.midtop = (x, y)
    surf.blit(text_surface, text_rect)


def draw_shield_bar(surf, x, y, pct):
    # if pct < 0:
    #     pct = 0
    pct = max(pct, 0)
    ## moving them to top
    # BAR_LENGTH = 100
```

```
    # BAR_HEIGHT = 10
    fill = (pct / 100) * BAR_LENGTH
    outline_rect = pygame.Rect(x, y, BAR_LENGTH, BAR_HEIGHT)
    fill_rect = pygame.Rect(x, y, fill, BAR_HEIGHT)
    pygame.draw.rect(surf, GREEN, fill_rect)
    pygame.draw.rect(surf, WHITE, outline_rect, 2)


def draw_lives(surf, x, y, lives, img):
    for i in range(lives):
        img_rect= img.get_rect()
        img_rect.x = x + 30 * i
        img_rect.y = y
        surf.blit(img, img_rect)



def newmob():
    mob_element = Mob()
    all_sprites.add(mob_element)
    mobs.add(mob_element)

class Explosion(pygame.sprite.Sprite):
    def __init__(self, center, size):
        pygame.sprite.Sprite.__init__(self)
        self.size = size
        self.image = explosion_anim[self.size][0]
        self.rect = self.image.get_rect()
        self.rect.center = center
        self.frame = 0
        self.last_update = pygame.time.get_ticks()
```

```
            self.frame_rate = 75

        def update(self):
            now = pygame.time.get_ticks()
            if now - self.last_update > self.frame_rate:
                self.last_update = now
                self.frame += 1
                if self.frame == len(explosion_anim[self.size]):
                    self.kill()
                else:
                    center = self.rect.center
                    self.image = explosion_anim[self.size][self.frame]
                    self.rect = self.image.get_rect()
                    self.rect.center = center


    class Player(pygame.sprite.Sprite):
        def __init__(self):
            pygame.sprite.Sprite.__init__(self)
            ## scale the player img down
            self.image = pygame.transform.scale(player_img, (145, 68))
            self.image.set_colorkey(BLACK)
            self.rect = self.image.get_rect()
            self.radius = 20
            self.rect.centerx = WIDTH / 2
            self.rect.bottom = HEIGHT - 10
            self.speedx = 0
            self.shield = 100
            self.shoot_delay = 250
            self.last_shot = pygame.time.get_ticks()
            self.lives = 3
            self.hidden = False
            self.hide_timer = pygame.time.get_ticks()
            self.power = 1
            self.power_timer = pygame.time.get_ticks()

        def update(self):
            ## time out for powerups
            if self.power >=2 and pygame.time.get_ticks() - self.power_time > POWERUP_TIME:
                self.power -= 1
                self.power_time = pygame.time.get_ticks()

            ## unhide
            if self.hidden and pygame.time.get_ticks() - self.hide_timer > 1000:
                self.hidden = False
                self.rect.centerx = WIDTH / 2
                self.rect.bottom = HEIGHT - 30

            self.speedx = 0     ## makes the player static in the screen by default.
            # then we have to check whether there is an event hanlding being done for the arrow keys being
            ## pressed

            ## will give back a list of the keys which happen to be pressed down at that moment
            keystate = pygame.key.get_pressed()
            if keystate[pygame.K_LEFT]:
                self.speedx = -5
            elif keystate[pygame.K_RIGHT]:
                self.speedx = 5

            #Fire weapons by holding spacebar
            if keystate[pygame.K_SPACE]:
                self.shoot()
```

```
187
188             ## check for the borders at the left and right
189             if self.rect.right > WIDTH:
190                 self.rect.right = WIDTH
191             if self.rect.left < 0:
192                 self.rect.left = 0
193
194             self.rect.x += self.speedx
195
196         def shoot(self):
197             ## to tell the bullet where to spawn
198             now = pygame.time.get_ticks()
199             if now - self.last_shot > self.shoot_delay:
200                 self.last_shot = now
201                 if self.power == 1:
202                     bullet = Bullet(self.rect.centerx, self.rect.top)
203                     all_sprites.add(bullet)
204                     bullets.add(bullet)
205                     shooting_sound.play()
206                 if self.power == 2:
207                     bullet1 = Bullet((self.rect.left+33), self.rect.centery)
208                     bullet2 = Bullet((self.rect.right-41), self.rect.centery)
209                     all_sprites.add(bullet1)
210                     all_sprites.add(bullet2)
211                     bullets.add(bullet1)
212                     bullets.add(bullet2)
213                     shooting_sound.play()
214
215                 if self.power >= 3:
216                     bullet1 = Bullet((self.rect.left+33), self.rect.centery)
217                     bullet2 = Bullet((self.rect.right-41), self.rect.centery)
218                     missile1 = Missile((self.rect.centerx-4), self.rect.top) # Missile shoots from center of ship
219                     all_sprites.add(bullet1)
220                     all_sprites.add(bullet2)
221                     all_sprites.add(missile1)
222                     bullets.add(bullet1)
223                     bullets.add(bullet2)
224                     bullets.add(missile1)
225                     shooting_sound.play()
226                     missile_sound.play()
227
228         def powerup(self):
229             self.power += 1
230             self.power_time = pygame.time.get_ticks()
231
232         def hide(self):
233             self.hidden = True
234             self.hide_timer = pygame.time.get_ticks()
235             self.rect.center = (WIDTH / 2, HEIGHT + 200)
236
237
238     # defines the enemies
239     class Mob(pygame.sprite.Sprite):
240         def __init__(self):
241             pygame.sprite.Sprite.__init__(self)
242             self.image_orig = random.choice(meteor_images)
243             self.image_orig.set_colorkey(BLACK)
244             self.image = self.image_orig.copy()
245             self.rect = self.image.get_rect()
246             self.radius = int(self.rect.width *.90 / 2)
247             self.rect.x = random.randrange(0, WIDTH - self.rect.width)
248             self.rect.y = random.randrange(-150, -100)
```

```
        self.speedy = random.randrange(5, 20)        ## for randomizing the speed of the Mob

        ## randomize the movements a little more
        self.speedx = random.randrange(-3, 3)

        ## adding rotation to the mob element
        self.rotation = 0
        self.rotation_speed = random.randrange(-8, 8)
        self.last_update = pygame.time.get_ticks()  ## time when the rotation has to happen

    def rotate(self):
        time_now = pygame.time.get_ticks()
        if time_now - self.last_update > 50: # in milliseconds
            self.last_update = time_now
            self.rotation = (self.rotation + self.rotation_speed) % 360
            new_image = pygame.transform.rotate(self.image_orig, self.rotation)
            old_center = self.rect.center
            self.image = new_image
            self.rect = self.image.get_rect()
            self.rect.center = old_center

    def update(self):
        self.rotate()
        self.rect.x += self.speedx
        self.rect.y += self.speedy
        ## now what if the mob element goes out of the screen

        if (self.rect.top > HEIGHT + 10) or (self.rect.left < -25) or (self.rect.right > WIDTH + 20):
            self.rect.x = random.randrange(0, WIDTH - self.rect.width)
            self.rect.y = random.randrange(-100, -40)
            self.speedy = random.randrange(1, 8)        ## for randomizing the speed of the Mob


## defines the sprite for Powerups
class Pow(pygame.sprite.Sprite):
    def __init__(self, center):
        pygame.sprite.Sprite.__init__(self)
        self.type = random.choice(['shield', 'gun'])
        self.image = powerup_images[self.type]
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        ## place the bullet according to the current position of the player
        self.rect.center = center
        self.speedy = 2

    def update(self):
        """should spawn right in front of the player"""
        self.rect.y += self.speedy
        ## kill the sprite after it moves over the top border
        if self.rect.top > HEIGHT:
            self.kill()



## defines the sprite for bullets
class Bullet(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = bullet_img
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        ## place the bullet according to the current position of the player
        self.rect.bottom = y
```

```
311             self.rect.centerx = x
312             self.speedy = -10
313 
314         def update(self):
315             """should spawn right in front of the player"""
316             self.rect.y += self.speedy
317             ## kill the sprite after it moves over the top border
318             if self.rect.bottom < 0:
319                 self.kill()
320 
321             ## now we need a way to shoot
322             ## lets bind it to "spacebar".
323             ## adding an event for it in Game loop
324 
325     ## FIRE ZE MISSILES
326     class Missile(pygame.sprite.Sprite):
327         def __init__(self, x, y):
328             pygame.sprite.Sprite.__init__(self)
329             self.image = missile_img
330             self.image.set_colorkey(BLACK)
331             self.rect = self.image.get_rect()
332             self.rect.bottom = y
333             self.rect.centerx = x
334             self.speedy = -10
335 
336         def update(self):
337             """should spawn right in front of the player"""
338             self.rect.y += self.speedy
339             if self.rect.bottom < 0:
340                 self.kill()
341 
```

```
342
343 ##################################################
344 ## Load all game images
345
346 background = pygame.image.load(path.join(img_dir, 'starfield.jpg')).convert()
347 background_rect = background.get_rect()
348 ## ^^ draw this rect first
349
350 player_img = pygame.image.load(path.join(img_dir, 'playerShip1_orange.png')).convert()
351 player_mini_img = pygame.transform.scale(player_img, (40, 23))
352 player_mini_img.set_colorkey(BLACK)
353 bullet_img = pygame.image.load(path.join(img_dir, 'laserRed16.png')).convert()
354 missile_img = pygame.image.load(path.join(img_dir, 'missile.png')).convert_alpha()
355 # meteor_img = pygame.image.load(path.join(img_dir, 'meteorBrown_med1.png')).convert()
356 meteor_images = []
357 meteor_list = [
358 ##     'meteorBrown_big1.png',
359 ##     'meteorBrown_big2.png',
360     'meteorBrown_med1.png',
361     'meteorBrown_med3.png',
362     'meteorBrown_small1.png',
363     'meteorBrown_small2.png',
364     'meteorBrown_tiny1.png'
365 ]
366
367 for image in meteor_list:
368     meteor_images.append(pygame.image.load(path.join(img_dir, image)).convert())
369
370 ## meteor explosion
371 explosion_anim = {}
372 explosion_anim['lg'] = []
373 explosion_anim['sm'] = []
374 explosion_anim['player'] = []
375 for i in range(9):
376     filename = 'regularExplosion0{}.png'.format(i)
377     img = pygame.image.load(path.join(img_dir, filename)).convert()
378     img.set_colorkey(BLACK)
379     ## resize the explosion
380     img_lg = pygame.transform.scale(img, (75, 75))
381     explosion_anim['lg'].append(img_lg)
382     img_sm = pygame.transform.scale(img, (32, 32))
383     explosion_anim['sm'].append(img_sm)
384
385     ## player explosion
386     filename = 'sonicExplosion0{}.png'.format(i)
387     img = pygame.image.load(path.join(img_dir, filename)).convert()
388     img.set_colorkey(BLACK)
389     explosion_anim['player'].append(img)
390
391 ## load power ups
392 powerup_images = {}
393 powerup_images['shield'] = pygame.image.load(path.join(img_dir, 'shield_gold.png')).convert()
394 powerup_images['gun'] = pygame.image.load(path.join(img_dir, 'bolt_gold.png')).convert()
395
396
397 ##################################################
398
399
400 ##################################################
401 ### Load all game sounds
402 shooting_sound = pygame.mixer.Sound(path.join(sound_folder, 'pew.wav'))
403 missile_sound = pygame.mixer.Sound(path.join(sound_folder, 'rocket.ogg'))
```

```
expl_sounds = []
for sound in ['expl3.wav', 'expl6.wav']:
    expl_sounds.append(pygame.mixer.Sound(path.join(sound_folder, sound)))
## main background music
#pygame.mixer.music.load(path.join(sound_folder, 'tgfcoder-FrozenJam-SeamlessLoop.ogg'))
pygame.mixer.music.set_volume(0.2)      ## simmered the sound down a little

player_die_sound = pygame.mixer.Sound(path.join(sound_folder, 'rumble1.ogg'))
################################################

## TODO: make the game music loop over again and again. play(loops=-1) is not working
# Error :
# TypeError: play() takes no keyword arguments
#pygame.mixer.music.play()

############################
## Game loop
running = True
menu_display = True
while running:
    if menu_display:
        main_menu()
        pygame.time.wait(3000)

        #Stop menu music
        pygame.mixer.music.stop()
        #Play the gameplay music
        pygame.mixer.music.load(path.join(sound_folder, 'tgfcoder-FrozenJam-SeamlessLoop.ogg'))
        pygame.mixer.music.play(-1)     ## makes the gameplay sound in an endless loop

        menu_display = False

        ## group all the sprites together for ease of update
        all_sprites = pygame.sprite.Group()
        player = Player()
        all_sprites.add(player)

        ## spawn a group of mob
        mobs = pygame.sprite.Group()
        for i in range(8):      ## 8 mobs
            # mob_element = Mob()
            # all_sprites.add(mob_element)
            # mobs.add(mob_element)
            newmob()

        ## group for bullets
        bullets = pygame.sprite.Group()
        powerups = pygame.sprite.Group()

        #### Score board variable
        score = 0

    #1 Process input/events
    clock.tick(FPS)     ## will make the loop run at the same speed all the time
    for event in pygame.event.get():        # gets all the events which have occured till now and keeps tab of them.
        ## listening for the the X button at the top
        if event.type == pygame.QUIT:
            running = False

        ## Press ESC to exit game
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
```

```
                running = False
            # ## event for shooting the bullets
            # elif event.type == pygame.KEYDOWN:
            #     if event.key == pygame.K_SPACE:
            #         player.shoot()      ## we have to define the shoot()  function

    #2 Update
    all_sprites.update()


    ## check if a bullet hit a mob
    ## now we have a group of bullets and a group of mob
    hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
    ## now as we delete the mob element when we hit one with a bullet, we need to respawn them again
    ## as there will be no mob_elements left out
    for hit in hits:
        score += 50 - hit.radius         ## give different scores for hitting big and small metoers
        random.choice(expl_sounds).play()
        # m = Mob()
        # all_sprites.add(m)
        # mobs.add(m)
        expl = Explosion(hit.rect.center, 'lg')
        all_sprites.add(expl)
        if random.random() > 0.9:
            pow = Pow(hit.rect.center)
            all_sprites.add(pow)
            powerups.add(pow)
        newmob()        ## spawn a new mob

    ## ^^ the above loop will create the amount of mob objects which were killed spawn again
    ########################
```

```

    ## check if the player collides with the mob
    hits = pygame.sprite.spritecollide(player, mobs, True, pygame.sprite.collide_circle)
    for hit in hits:
        player.shield -= hit.radius * 2
        expl = Explosion(hit.rect.center, 'sm')
        all_sprites.add(expl)
        newmob()
        if player.shield <= 0:
            player_die_sound.play()
            death_explosion = Explosion(player.rect.center, 'player')
            all_sprites.add(death_explosion)
            # running = False     ## GAME OVER 3:D
            player.hide()
            player.lives -= 1
            player.shield = 100

    ## if the player hit a power up
    hits = pygame.sprite.spritecollide(player, powerups, True)
    for hit in hits:
        if hit.type == 'shield':
            player.shield += random.randrange(10, 30)
            if player.shield >= 100:
                player.shield = 100
        if hit.type == 'gun':
            player.powerup()

    ## if player died and the explosion has finished, end game
    if player.lives == 0 and not death_explosion.alive():
        running = True
        menu_display = True
```

```
516         for hit in hits:
517             if hit.type == 'shield':
518                 player.shield += random.randrange(10, 30)
519                 if player.shield >= 100:
520                     player.shield = 100
521             if hit.type == 'gun':
522                 player.powerup()
523 
524         ## if player died and the explosion has finished, end game
525         if player.lives == 0 and not death_explosion.alive():
526             running = True
527             menu_display = True
528             pygame.display.update()
529 
530         #3 Draw/render
531         screen.fill(BLACK)
532         ## draw the stargaze.png image
533         screen.blit(background, background_rect)
534 
535         all_sprites.draw(screen)
536         draw_text(screen, str(score), 18, WIDTH / 2, 10)     ## 10px down from the screen
537         draw_shield_bar(screen, 5, 5, player.shield)
538 
539         # Draw lives
540         draw_lives(screen, WIDTH - 100, 5, player.lives, player_mini_img)
541 
542         ## Done after drawing everything to the screen
543         pygame.display.flip()
544 
545     pygame.quit()
546 
```

➢ COMMENTS AND DESCRIPTION OF CODING SEGMENT:

Comments and description are given with the code in the above images.

➢ STANDARDIZATION OF THE CODING:

This section can be the most knowledgeable section as the whole project can be defined under this section.

1. Importing the modules:

Firstly, we have started the program and imported all the necessary modules.

2. Opening of the new window:

Then, defined the open the new window using blit and redraw function, setting the window's width and height and is also responsible for the pop up of the new screen.

3. Initialization of the sprites:

In addition to this, we have initialized the various sprites used in this game. Sprites like variable, health, shields, lasers, etc. In this section, we have loaded the player ship's image and background images followed by the laser's image etc. up to 1 to 3, they all will run only once, when we launch the game.

4. Main Program:

Then comes the main executive loop, i.e., main program while loop. It repeats itself x number of times per second being the framerates.

5. Capture events:

Then comes the event capturing (e.g., user interaction) where the start, quit, move all occurs when the user clicks on the screen, or presses the key on the keyboard. It actually confirms also that rather the user has clicked the keyboard or not?

6. Game Logic:

In this coding section, the logic of the game has been used. Like what happens between two frames? (e.g., position of ship)

7. Refresh screen:

It redraws the background and all the sprites.

➢ CODE EFFICIENCY:

It is actually a term used to analyze the reliability, speed and programming which have been used for developing codes for an application. We have ensured that this game would run in every type of PC having higher and slower running speed due to the usage of FPS which syncs with the clock time, maintaining the consistency of the game.

```
33  ## initialize pygame and create window
34  pygame.init()
35  pygame.mixer.init()  ## For sound
36  screen = pygame.display.set_mode((WIDTH, HEIGHT))
37  pygame.display.set_caption("Space Shooter")
38  clock = pygame.time.Clock()     ## For syncing the FPS
39  ##############################
```

➢ ERROR HANDLING:

The customized and specialized coding or programs are called the error handling usually. It is the detection, application, communication resolution errors which can occur in logic or syntax. There were various errors encountered like desync of firing and audio, function definitions or compile time error and much more. But we tried our best to resolve it using various website forums like GitHub / Stack Overflow / Reddit etc.

Some of the errors encountered were:

1. Runtime Errors due to spellings and syntax errors:

The errors encountered due to syntactical error or spelling mistakes are considered as Runtime Errors. To solve these errors, we found all the syntactical and spelling mistakes in the coding section and solved it efficiently using the correct syntax.

Example:

Python

```
def some_function()
    msg = 'hello, world!'
    print(msg)
     return msg
```

Error

```
  File "<ipython-input-3-6bb841ea1423>", line 1
    def some_function()
                       ^
SyntaxError: invalid syntax
```

Python

```
def some_function():
    msg = 'hello, world!'
    print(msg)
     return msg
```

Error

```
  File "<ipython-input-4-ae290e7659cb>", line 4
    return msg
    ^
IndentationError: unexpected indent
```

2. Desync of audio and visual elements (firing and audio):

There were various errors due to desync between the audio files and the visual elements (firing, explosions, lasers). To solve these desyncs between the audio and visual elements, we changed the timing or the total during the sound were being played for, by incrementing and decrementing the total audio or visual time limits till they were working in synergy.

3. Improper calling of methods / variables:

We came across several errors for using wrong method names or wrong variables. To solve this, we corrected the specific variable names and methods so that the rest of the code is not altered by the same.

Example:

Python

```
print(a)
```

Error

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-9d7b17ad5387> in <module>()
----> 1 print(a)

NameError: name 'a' is not defined
```

Python

```
print(hello)
```

Error

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-9553ee03b645> in <module>()
----> 1 print(hello)

NameError: name 'hello' is not defined
```

Python

```
for number in range(10):
    count = count + number
print('The count is:', count)
```

Error

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-dd6a12d7ca5c> in <module>()
      1 for number in range(10):
----> 2     count = count + number
      3 print('The count is:', count)

NameError: name 'count' is not defined
```

Python

```
Count = 0
for number in range(10):
    count = count + number
print('The count is:', count)
```

Error

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-d77d40059aea> in <module>()
      1 Count = 0
      2 for number in range(10):
----> 3     count = count + number
      4 print('The count is:', count)

NameError: name 'count' is not defined
```

4. Exception errors:

Sometimes even if the expression or snippet is syntactically correct, the code may cause an error when we try to run it. Errors which are encountered during the runtime of a program is known as Exceptions. To fix these Exceptions, we thoroughly read through the program again and again, trying to find the exception and solved it using Exception Handling (try & except statement).

5. File errors:

Sometimes a file which is not found or which doesn't exist or which is causing an error during merging with other assets or the code can cause file errors while runtime. To tackle these, either we have to create or rectify the file path or we have to check the syntax or use another way to bind the file to the code.

Example:

Python

```
file_handle = open('myfile.txt', 'r')
```

Error

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-14-f6e1ac4aee96> in <module>()
----> 1 file_handle = open('myfile.txt', 'r')

FileNotFoundError: [Errno 2] No such file or directory: 'myfile.txt'
```

6. Index errors:

Certain errors occur when we try to call an index or use it, when it doesn't exist in the program. To solve these, we created the correct indexes which were being called by us, or we corrected the existing indexes for the same.

Example:

Python

```
letters = ['a', 'b', 'c']
print('Letter #1 is', letters[0])
print('Letter #2 is', letters[1])
print('Letter #3 is', letters[2])
print('Letter #4 is', letters[3])
```

Output

```
Letter #1 is a
Letter #2 is b
Letter #3 is c
```

Error

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-11-d817f55b7d6c> in <module>()
      3 print('Letter #2 is', letters[1])
      4 print('Letter #3 is', letters[2])
----> 5 print('Letter #4 is', letters[3])

IndexError: list index out of range
```

➢ PARAMETERS CALLING/PASSING:

Following are the methods used in the code, which are being called or passed:

1. **Join ():** It is being used to join or merge the directory and the files in which our assets or sounds are kept at.

```
## assets folder
img_dir = path.join(path.dirname(__file__), 'assets')
sound_folder = path.join(path.dirname(__file__), 'sounds')
```

2. **Init ():** It is the function or method, which can be compared with the constructor and is used to initialise the instance's or object's state.

```
pygame.init()
pygame.mixer.init()  ## For sound
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Space Shooter")
clock = pygame.time.Clock()     ## For syncing the FPS
################################

font_name = pygame.font.match_font('arial')
```

3. **Main_menu ():** This is the main executive loop of our coding where every methods are being called and defined

```
def main_menu():
    global screen

    menu_song = pygame.mixer.music.load(path.join(sound_folder, "menu.ogg"))
    pygame.mixer.music.play(-1)

    title = pygame.image.load(path.join(img_dir, "main.png")).convert()
    title = pygame.transform.scale(title, (WIDTH, HEIGHT), screen
    screen.blit(title, (0,0))
    pygame.display.update()
```

4. **Draw_text ():** In this section, the pop up or triggered message get displayed because of this method. Also in the other image, health, bar, lives and movements are passed under the draw_text method.

```
else:
    draw_text(screen, "Press [ENTER] To Begin", 30, WIDTH/2, HEIGHT/2)
    draw_text(screen, "or [Q] To Quit", 30, WIDTH/2, (HEIGHT/2)+40)
    pygame.display.update()
```

```
def draw_text(surf, text, size, x, y):
    ## selecting a cross platform font to display the score
    font = pygame.font.Font(font_name, size)
    text_surface = font.render(text, True, WHITE)       ## True denotes the font to be anti-aliased
    text_rect = text_surface.get_rect()
    text_rect.midtop = (x, y)
    surf.blit(text_surface, text_rect)
```

```
def draw_shield_bar(surf, x, y, pct):
```

```
def draw_lives(surf, x, y, lives, img):
```

```
def newmob():
    mob_element = Mob()
    all_sprites.add(mob_element)
    mobs.add(mob_element)
```

```
def __init__(self, center, size):
```

5. **Load method ():** In this section, we've simply loaded the images of the various sprites and backgrounds and passed the images only of the png format.

```
## Load all game images

background = pygame.image.load(path.join(img_dir, 'starfield.png')).convert()
background_rect = background.get_rect()
## ^^ draw this rect first

player_img = pygame.image.load(path.join(img_dir, 'playerShip1_orange.png')).convert()
player_mini_img = pygame.transform.scale(player_img, (25, 19))
player_mini_img.set_colorkey(BLACK)
bullet_img = pygame.image.load(path.join(img_dir, 'laserRed16.png')).convert()
missile_img = pygame.image.load(path.join(img_dir, 'missile.png')).convert_alpha()
# meteor_img = pygame.image.load(path.join(img_dir, 'meteorBrown_med1.png')).convert()
meteor_images = []
meteor_list = [
    'meteorBrown_big1.png',
    'meteorBrown_big2.png',
    'meteorBrown_med1.png',
    'meteorBrown_med3.png',
    'meteorBrown_small1.png',
    'meteorBrown_small2.png',
    'meteorBrown_tiny1.png'
]
```

```
## load power ups
powerup_images = {}
powerup_images['shield'] = pygame.image.load(path.join(img_dir, 'shield_gold.png')).convert()
powerup_images['gun'] = pygame.image.load(path.join(img_dir, 'bolt_gold.png')).convert()
```

```
### Load all game sounds
shooting_sound = pygame.mixer.Sound(path.join(sound_folder, 'pew.wav'))
missile_sound = pygame.mixer.Sound(path.join(sound_folder, 'rocket.ogg'))
expl_sounds = []
for sound in ['expl3.wav', 'expl6.wav']:
    expl_sounds.append(pygame.mixer.Sound(path.join(sound_folder, sound)))
## main background music
#pygame.mixer.music.load(path.join(sound_folder, 'tgfcoder-FrozenJam-SeamlessLoop.ogg'))
pygame.mixer.music.set_volume(0.2)      ## simmered the sound down a little

player_die_sound = pygame.mixer.Sound(path.join(sound_folder, 'rumble1.ogg'))
```

6. **Group ():** I n this method, we've grouped all the sprites together for ease of update

```
## group all the sprites together for ease of update
all_sprites = pygame.sprite.Group()
player = Player()
all_sprites.add(player)
```

```
## group for bullets
bullets = pygame.sprite.Group()
powerups = pygame.sprite.Group()
```

7. **Wait() :** After every 3000 second, the game will run until it's true. Hence the value has been passed as 3000 under the wait function.

```
pygame.time.wait(3000)
```

8. **Play ():** -1 has been passed in the play function which makes the gameplay sound in an endless loop until it's over.

```
#Play the gameplay music
pygame.mixer.music.load(path.join(sound_folder, 'tgfcoder-FrozenJam-SeamlessLoop.ogg'))
pygame.mixer.music.play(-1)     ## makes the gameplay sound in an endless loop
```

9. **Tick ():** This function where FPS has been passed will make the loop run at the same speed all the time in any device. Hence, will make the game consistent.

```
clock.tick(FPS)     ## will make the loop run at the same speed all the time
for event in pygame.event.get():        # gets all the events which have occured till now and keeps tab of them.
```

10. **Groupcollide ():** It is the function where mobs, bullets have been passed which helps in grouping the mobs and bullets.

```
## now we have a group of bullets and a group of mob
hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
```

11. **Explosion ():** This function is responsible for the explosion.

```
expl = Explosion(hit.rect.center, 'sm')
all_sprites.add(expl)
newmob()
if player.shield <= 0:
    player_die_sound.play()
    death_explosion = Explosion(player.rect.center, 'player')
    all_sprites.add(death_explosion)
```

12. **Blit ():** This is the function where the blitting of the image is responsible. It covers the upper left corner of the screen which is almost equal to the ¼ th of the screen. Hence, to make it suitable, blit () has been used.

```
screen.fill(BLACK)
## draw the stargaze.png image
screen.blit(background, background_rect)

all_sprites.draw(screen)
draw_text(screen, str(score), 18, WIDTH / 2, 10)     ## 10px down from the screen
draw_shield_bar(screen, 5, 5, player.shield)

# Draw lives
draw_lives(screen, WIDTH - 100, 5, player.lives, player_mini_img)

## Done after drawing everything to the screen
pygame.display.flip()
```

➢ VALIDATION CHECKS:

Validation checks: It is fully validation checked; hence it doesn't provide the third-party user to access or mishandle our code intentionally accidentally. Although we've tried to provide our users to give the feedbacks that would be helpful for us to enhance the game furthermore.

The use of this section is to simply check if the input data type is correct or not and it can be checked if there are no valid values in the input data.

Ch 5. TESTING

This document describes the design of space shooter. It contains the requirements, theory of operation and dependencies. Schematic diagrams, and code are used to demonstrate the architecture described. A list of protocol used to test functionality of the Space Shooters are explained. The mechanical and hardware description of the Space Shooter is not included.

➢ TESTING TECHNIQUES AND TESTING STRATEGIES USED:

Testing determines whether the code runs correctly and identifies the defects that need to be fixed for running the code. For testing we use several techniques and strategies which are covered in this section.

We have used different ways to test our coding like:

- **Unit Testing:**

Unit Testing is the first phase of testing so we used this procedure first. In this method, the developers make sure that every individual component of a software is working as they were designed and coded for and fulfilling their functionality. This process also makes the debugging process easier. This testing can be done by the developers themselves.

- **Integration Testing:**

Integration Testing makes sure that each and every function / method are behaving as expected from them and are not causing any errors for the rest of the code. This testing can be done by developers or independent testers and it also consists of some of the manual testing in it, which are mentioned below.

- **System Testing:**

System Testing is a test which is used on fully integrated and fully completed and developed software. It is conducted in a black box, which ensures and focuses on the functionality of the software. In this testing all the user inputs are tested for various test cases. This test is usually conducted by the Quality Analysis team, but we managed to do this somehow on our own.

- **Performance Testing:**

Performance Testing is conducted to see how different components are behaving under several conditions when encountered face to face with them.
This testing ensures the responsiveness and stability of the software while making simulations for the real gameplay by any user.

This testing is further divided into several parts:

1. Load Testing:

Load Testing is conducted to check the benchmarks or the full performance potential of the software and to test how much load by the CPU and user it takes until its failure.

2. Stress Testing:

Stress Testing is one level above the Load Testing as, in the testing we test how the software responds and behaves beyond the point of its loading capacity.

3. Endurance Testing:

This testing is also known as Soak Testing, and it is used for analyzing the behaviour of a software for a specific period of fixed time under which it is being simulated for pure performance testing. This test usually finds out the memory leaks in a software, but luckily, we didn't get any of the memory leaks during the testing.

4. Spike Testing:

Spike Testing is a type of Load Testing where the software is tested for its responsiveness while being exposed to simulated performance testing in longer bursts of time for various number of times. This shows how the software behaves under sudden high usage and low usage time.

- **Compatibility Testing:**

Compatibility Testing is conducted for finding out the responsiveness and analyzing the behaviour of the software when it is being used in multiple operating systems such as windows, macOS, Linux, android and much more.

- **Manual tests:**

In manual test we create different python file to test the code for particular purpose such as for home page, Game Loop, Moving the player ship across the axis.

- **IDE:**

For coding we used Visual Studio IDE (integrated development environment) which display the red mark on the error line while running the code.

➢ TESTING PLAN USED:

We used various Testing Plans for implementing the testing techniques for the project. Some of these Testing Plans are listed below:

1. **Analysing the product.**
2. **Defining the Test Objectives.**
3. **Defining the Test Criteria.**
4. **Resource Planning.**
5. **Planning the Test Environment.**
6. **Scheduling.**
7. **Determining Test Deliverables.**

➢ DEBUGGING AND CODE IMPROVEMENT:

Debugging is the procedure of finding and rectifying the errors/ defects / problems within a programming code that prevents the program from being executed in the correct or systematical way.

We used the following debugging methods for debugging the program:

1. Brute Force Method:

Brute Force Method is one of the most common practices used for debugging the code, which is the least time and power consuming. In this method, the program is introduced with various print statements so that while executing the program we can see the exact number of times the print statement is being executed and in which section the error is being introduced at, for the effective and exact debugging of the code.

2. Backtracking:

In Backtracking Method, the source code is searched for errors and bugs from the exact time the error symptom was found at and backtracked backwards till the exact error is introduced. But the time and energy consumed by this method is extremely huge, so we decided to debug the code once or twice only, using this method.

3. Cause Elimination Method:

In Cause Elimination Method, the developers try to imitate and introduce an exact type of error scenario, to understand the error better and to rectify the problem from its roots.
So basically, we find an error during runtime, if we don't understand the source of the problem, we try to induce a similar error of our own, and then deduce the error by understanding the exact reason for the error. This is why this method is also known as Induction and Deduction.

4. Program Slicing:

Program Slicing in similar to the Backtracking Method, but the difference is, that the code is divided into several different slices and then debugging is conducted on them. It is way more efficient than Backtracking method in terms of time expenditure and energy consumption. It is also an easy process of finding errors more easily.

➢ SYSTEM SECURITY MEASURES (IMPLEMENTATION OF SECURITY FOR THE PROJECT DEVELOPED):

Since this project was just for educational purpose and a small-scale project, we did not require any type of security measures for the project development.

Instead, we decided to send the project files and assets in our own private discord server and to store our files in google drives, which uses Two Factor Authentication (2FA) to protect the user accounts, and supports user end to end encryption for chats and file.

Therefore, there was no additional need of any more security measures to opt for.

➢ DATABASE/DATA SECURITY:

Since we made our project on the basis of Water fall model and Secure development Lifecycle, thus they offer various data security for our project. Some of these security features are:

1. Higher Security:

Due to Secure Development Lifecycle, there is continuous monitoring for numerous possibilities for vulnerabilities, resulting in a safe and secure software.

2. Regulatory Compliance:

Secure Development Lifecycle consists of high security related laws which has to be followed when following this lifecycle. Thus, this ensures the security of the data in the software.

3. Security Requirements:

There is a bar of security set for the software using the Secure Development Lifecycle. If the software fails to pass that bar, then there can be some security related issues in the software. Our project is based on Secure Development Lifecycle, and ensures data security.

4. Approved tools:

Since this project is constructed using Secure Development Lifecycle approved tools and strategies, thus there can be no data security breaches.

➢ CREATION OF USER PROFILES AND ACCESS RIGHTS:

All our project related data were kept in Google Drives and shared in discord when in need or use. Hence the access rights were only shared with the project team i.e., Anchal, Rajat and Debashish only.

There was no need of creation of new user profiles since we already had our own Google and Discord accounts / profiles.

Ch 6. DOCUMENTATION

➢ COST ESTIMATION OF THE PROJECT ALONG WITH COST ESTIMATION MODEL:

Since this project is being made for experience and to gain knowledge and we found all the necessary resources and technologies required for the game for free (open-source), therefore there was no investment made in the form of capital and there was no Cost Estimation required for the project and same for the Cost Estimation Model.

➢ PRE- REQUISITE FOR PROJECT INSTALLATION:

Minimum Requirements:
OS: Linux/Ubuntu/Windows XP/ 7/ 8/ 10/ 11 (32-bit OS or more)
Processor: Intel Pentium (1.9 GHz) or greater
Memory: 1 GB RAM
Graphics: 64 MB
DirectX: Version 8 or more
Storage: 5 MB

➢ COMPARATIVE ANALYSIS:

Space Invaders (1978):
- Pixel game graphics used.
- Made for Arcade gaming only.
- Made for high end devices like arcade machines.
- Controls were only operable using the joystick.

Space Shooter (our project):
- New game graphics used.
- Made for laptops, PCs and console.
- Made in consideration of low-end devices.
- Control inputs are possible using mouse and keyboard.

➢ NEGATIVE RESULTS:

This project doesn't consist of any Negative results, either environmental or in terms of hardware except spending long during of time playing the game, which can be harmful for the eyes.

Ch 7. CONCLUSION AND FUTURE SCOPE

➢ CONCLUSION:

This game was created while keeping the things in mind that the project is created for educational purpose and for entertainment of the user, where the main motive of the player is to survive the incoming waves of meteors as long as possible. While making the game, we faced various problems in the coding part, had to find suitable graphics and sounds for the game, merge them systematically and fix the bugs in the programming part. During the making of project, we learned various things as how to efficiently work in a team, improved our python knowledge and how to debug the code. Despite all the difficulties, the game was successfully created by our team.

➢ FUTURE SCOPE AND FURTHER ENHANCEMENT OF THE PROJECT:

The game can be still enhanced depending on the requirements of the user such as by adding a Hall of Fame at the End of the game, adding more interactive HUD, more interactive and moving graphics, adding multiplayer mode and much more.

➢ WORK PROGRESS AS TIMELINE:

We have done our project up to the required goal. We have designed, coded and planned accordingly to the timeline.

Work done during First Report Submission (15/8/21 – 20/9/21):

- Installed the required software for the project. (Python, Pygame, Environment etc.)
- Created the game overlay. (Game window and basic functions)
- Created and also downloaded and edited the game graphics used.
- Researched for respective sound effects for the game.
- Researched for various fonts to be used.
- Tested for Plagiarism.

Work done during Second Report Submission (21/9/21 – 15/10/21):

- Added various new functions to the code. (for shooting, redrawing meteors, meteor behavior, health, powerups etc.)
- Merged the game graphics and sound effects in the code.
- Change in game graphics.
- Tested the game for bugs/ errors.
- Tested for Plagiarism.

Work done during Final Report Submission (16/10/21 – 5/11/21):

- Minor changes in the game UI.
- Updated the HUD and graphics.
- Updated the game as per user feedback and requirements.
- Tested the game for Quality and bugs/ errors.
- Tested for Plagiarism.

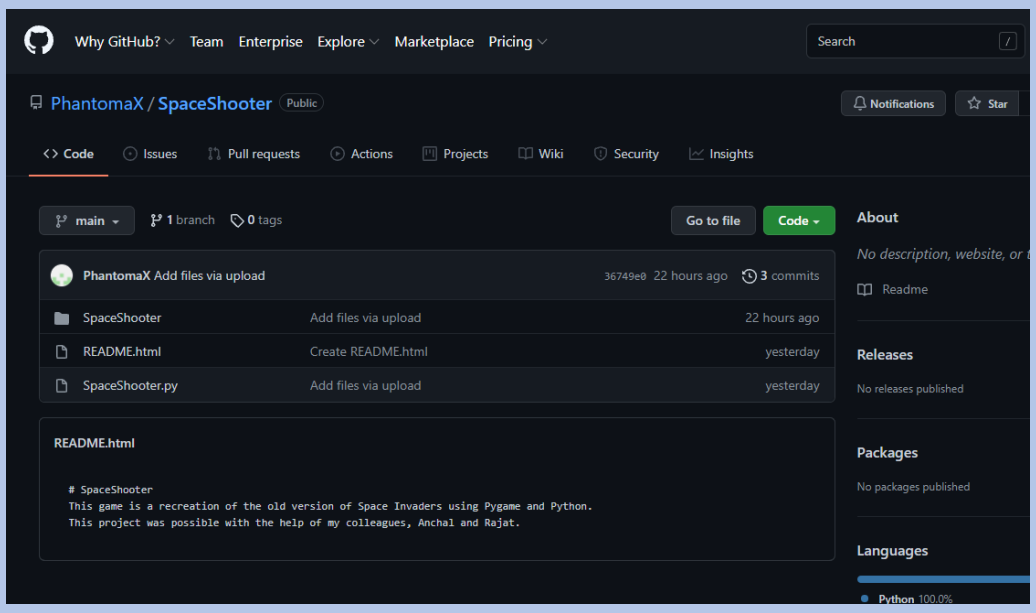
➢ EXPECTED OUTCOME:

The outcome till now has stood quite good as per we expected. Sounds, movements, lighting, 3D effects all are appealing. As per the given time, we can say that we have done simply well. Still, we need more time to make it more enhancing and advanced and for this we need some quite well clocks for doing the same.

We believe that this game has most of the important features that make up an enjoyable game. Some of the features are explosion effects, sound effects and a good visual environment. Needless to say, however, that there is a lot of other features that can be added and further improvements that can be made.

It must also be said that this implementation of "Space Invaders" is very much different from the original version, both visually and in the way it is played. Therefore, though the game is designed from the idea of "Space Invaders", we believe that this version has more than enough differences to qualify as a new game that uses some of the elements of the original version.

PROJECT SUBMISSION IN GITHUB



The Project is uploaded in a GitHub repository. This is the screenshot of GitHub repository where the account is public and can be accessed by everyone.

Link for the same is: https://github.com/PhantomaX/SpaceShooter

PLAGIARISM REPORT

We are fully concerned about the principle of plagiarism and so we have made this project with our conscious mind. Some terms, words are there which can be detected under the plagiarism as those are the heading or words which can't be customized, hence we decided to keep them as it is. Those terms are from the **provided format for certificates, acknowledgement page, table of index, pre requisite page and some basic words which cannot be changed**. On the other hand, we confidently assure you that it can't detect any plagiarism because we ourselves have checked the report twice from the verified websites. Hence, we believe that our marks won't get deducted in this kind of issues.

Plagiarism check is done from the following sites:
https://www.duplichecker.com/
https://www.check-plagiarism.com/

BIBLIOGRAPHY

Markowitz M. (2021, September 17), Space Invaders:
https://en.wikipedia.org/wiki/Space_Invaders

(2021, November 1), Pygame:
https://www.pygame.org/docs/ **(for finding various Pygame and Python commands)**

(2021, November 5), Stack overflow:
https://stackoverflow.com/ **(for coding help, error handling and debugging process)**

(2021, November 3), Reddit:
https://www.reddit.com/ **(for coding help, error handling and debugging process)**

(2021, November 5), Open Game Art:
https://opengameart.org/ **(for finding game graphics)**

(2021, November 5), Find Sounds:
https://www.findsounds.com/ **(for finding public open-source audio for game)**

(2021, November 4), Lucid chart:
https://www.lucidchart.com/ **(for making ER diagrams, DFD etc.)**

(2021, November 1), E Draw Max:
https://www.edrawmax.com/ **(for drawing PERT and GANTT chart)**

(2021, November 5), Check Plagiarism:
https://www.check-plagiarism.com/ **(for checking plagiarism)**

(2021, November 5), Dupli Checker:
https://www.duplichecker.com/ **(for checking plagiarism)**