BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:	**Surname**	**Given Names**	**Student ID Number**
1.	**Ihsan Kamal**	**Fauzan**	**2101720700**

Course Code	**: COMP6502**	**Course Name**	**: Introduction to Programming**
Class	**: L1BC**	**Name of Lecturer(s)**	**:** 1. Minaldi Loeis 2. Jude Martinez
Major	**: Computer Science**		
Title of Assignment (if any)	: Space Battle		
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 8-11-2017**	**Submission Date**	**: 8-11-2017**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.
Signature of Student:

Fauzan Ihsan Kamal

“Space Battle”

Name = Fauzan Ihsan Kamal

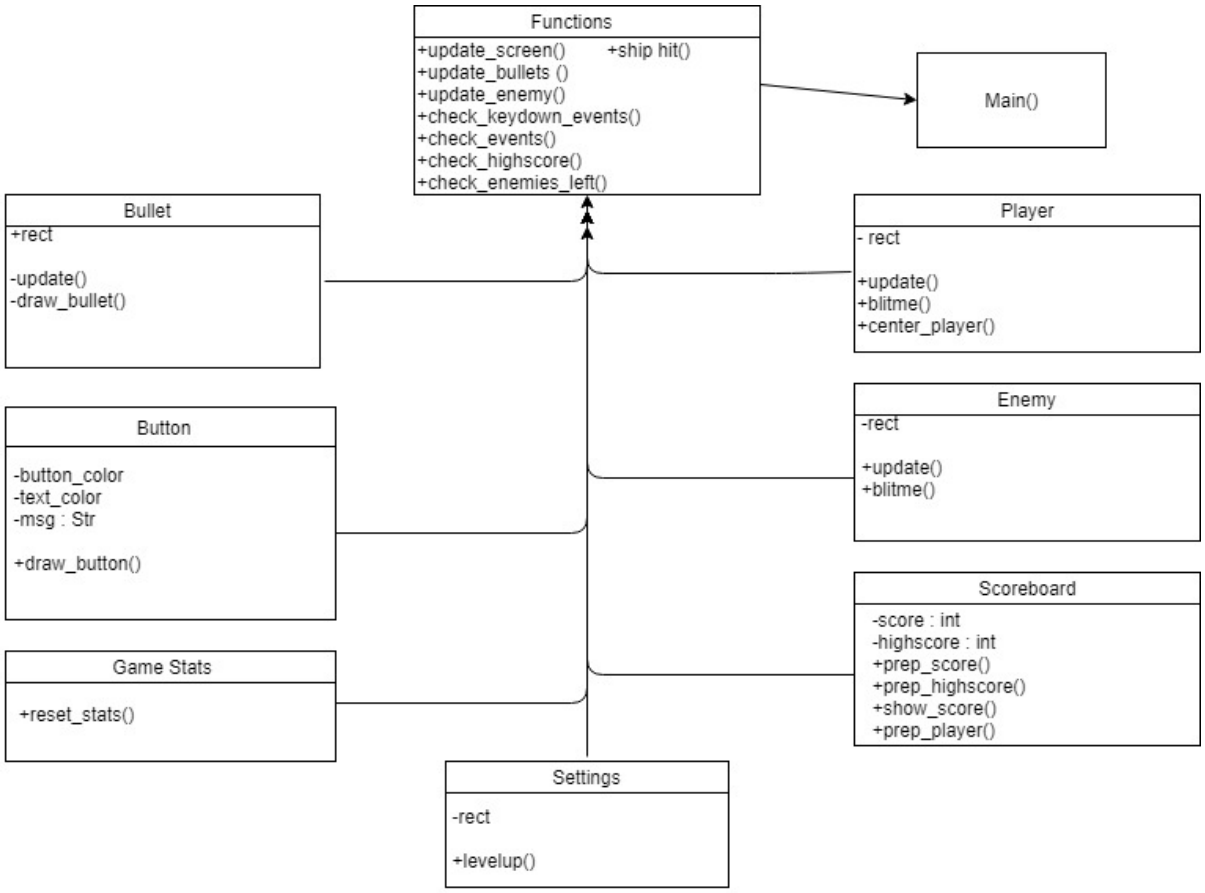
ID = 2101720700

I. Description

The function of this program :

This program is meant for the people who are bored and want to play a 2D endless shooter game on their computer. This game takes place in space where the player must shoot the aliens as much as they can to survive and gain scores and the difficulty is based on the player’s score. The higher the score, the more difficult the game gets. The game ends when an alien hits the player’s ship or reaches the edge of the left screen. This game is based on alien invasion game from python crash course.

II. UML Diagram



Text

III. Explanation of each function

Class Player (*player.py*)

- The purpose of this class is to adjust the player's position and also the player's movement.
- In *def update()* , I adjust the player's movement and also limits the movement. So the player cant get pass through the edge of the screen.
- In *def center_player()*, I input the player's position when the game starts or when the player respawns.
- *def blime* is to draw the player's ship into the screen

```
4   class Player(Sprite):
5       def __init__(self,ai_settings, screen):
6           super(Player, self).__init__()
7           self.screen = screen
8           self.ai_settings = ai_settings
9
10          self.image = pygame.image.load('images/futuramaship.png')
11          self.rect = self.image.get_rect()
12          self.screen_rect = screen.get_rect()
13
14
15          self.rect.centery = self.screen_rect.centery
16          self.rect.left = self.screen_rect.left
17          self.center = float(self.rect.centery)
18
19
20          self.moving_down = False
21          self.moving_up = False
22          self. moving_right = False
23          self.moving_left = False
24
25
26      def update(self):
27          if self.moving_down and self.center <= self.screen_rect.bottom:
28              self.center += self.ai_settings.player_speed_factor
29
30          if self.moving_up and self.center >= self.screen_rect.top:
31              self.center -= self.ai_settings.player_speed_factor
32
33          if self.moving_right and self.rect.right <= self.screen_rect.right:
34              self.rect.left += self.ai_settings.player_speed_factor
```

Player › __init__()

```
21          self.moving_up = False
22          self. moving_right = False
23          self.moving_left = False
24
25
26      def update(self):
27          if self.moving_down and self.center <= self.screen_rect.bottom:
28              self.center += self.ai_settings.player_speed_factor
29
30          if self.moving_up and self.center >= self.screen_rect.top:
31              self.center -= self.ai_settings.player_speed_factor
32
33          if self.moving_right and self.rect.right <= self.screen_rect.right:
34              self.rect.left += self.ai_settings.player_speed_factor
35
36          if self.moving_left and self.rect.left >= self.screen_rect.left:
37              self.rect.left -= self.ai_settings.player_speed_factor
38
39          self.rect.centery = self.center
40
41
42
43      def blitme(self):
44          self.screen.blit(self.image, self.rect)
45
46      def center_player(self):
47          self.center = self.screen_rect.centery
48          self.rect.left = self.screen_rect.left
49
50
51
```

Player › __init__()

Class Enemy (*enemy.py)*

- The purpose of this class is to adjust the enemy's position when it spawns and to set the enemy's direction when it moves
- For the position, I use *self.rect.y = random.randint(0,500)* to make the enemy spawn randomly in the y position between 0 and 500.
- In *def update()*, I set it -= so that the enemy can move to the left from the edge of the right screen.
- *def blitme()* is to draw the enemy

```
class Enemy(Sprite):
    """a class to represent a single enemy"""
    def __init__(self, ai_settings, screen):
        super(Enemy, self).__init__()
        self.screen = screen
        self.ai_settings = ai_settings


        self.image = pygame.image.load('images/cartoon-spaceship1.png')
        self.rect = self.image.get_rect()
        self.rect.right = 1300


        self.rect.y = random.randint(0,500)

        self.x = float(self.rect.x)
        self.y = float(self.rect.y)

    #to draw the enemy
    def blitme(self):
        self.screen.blit(self.image, self.rect)

    #to set the direction of enemy
    def update(self):
        self.rect.right -= self.ai_settings.enemy_speed_factor + self.ai_settings.level
```

Enemy › blitme()

Class Bullet (*bullet.py*)

- This class is to adjust the position and the direction of the player's bullet.
- In *def update()* , I adjust the speed and the position of the bullet
- *def draw_bullet* is to draw the bullet into the screen

```
import pygame
from pygame.sprite import Sprite

class Bullet(Sprite):
    def __init__(self, ai_settings, screen, player):
        super(Bullet, self).__init__()
        self.screen = screen

        self.rect = pygame.Rect(0,0,ai_settings.bullet_width, ai_settings.bullet_height)
        self.rect.centerx = player.rect.centerx
        self.rect.centery = player.rect.centery

        self.x = float(self.rect.x)
        self.color = ai_settings.bullet_color


    def update(self):
        self.x += 10
        self.rect.x = self.x


    def draw_bullet(self):
        pygame.draw.rect(self.screen, self.color, self.rect)
```

Bullet

Class Settings (*settings.py*)

- This class stores all of the game's settings (screen, level, player, enemy, etc) so I it makes the call code simpler and I can modify the settings easily.
- *def levelup* is to set the score on when the level changes

```python
class Settings():
    def __init__(self):
        #Screen settings
        self.screen_width = 1200
        self.screen_height = 600

        #level settings
        self.level = 1

        #player settings
        self.player_speed_factor = 7
        self.ship_limit = 0

        #enemy settings
        self.enemy_speed_factor = 7

        #bullet settings
        self.bullet_width = 16
        self.bullet_height = 3
        self.bullet_color = 255,0,0
        self.bullets_allowed = 20


        #scoring
        self.enemy_points = 50

    #function to level up each time player gain +500 score
    def levelup(self, score):
        if score > 500:
            self.level = 2
        if score > 1000:
```

Settings › __init__()

Class Button (*button.py*)

- The usage of this class is to make the play button appear before starting the game.
- *def prep_msg* is to turn the text into a rendered image and to center the text on the button.
- *def draw_button* is to draw the play button into the screen

```python
import pygame.font

#class to make the play button
class Button():
    def __init__(self, ai_settings, screen, msg):
        self.screen = screen
        self.screen_rect = screen.get_rect()

        self .width, self.height = 200, 50
        self.button_color = (255,0, 0)
        self.text_color = (255, 255, 255)
        self.font = pygame.font.SysFont(None, 48)

        self.rect = pygame.Rect(0, 0, self.width, self.height)
        self.rect.center = self.screen_rect.center

        self.prep_msg(msg)

    def prep_msg(self, msg):
        self.msg_image = self.font.render(msg, True, self.text_color,self.button_color)
        self.msg_image_rect = self.msg_image.get_rect()
        self.msg_image_rect.center = self.rect.center

    def draw_button(self):
        self.screen.fill(self.button_color, self.rect)
        self.screen.blit(self.msg_image, self.msg_image_rect)

```

Class Scoreboard (*scoreboard.py)*

- The purpose of this class is to make the score and the highscore appear in-game and also to adjust position of the score and the highscore.
- In *def prep_score*, I adjust the score's position at the top right of the screen and I also add *score_str* to add a coma when the score reach >= 1000.
- *def prep high_score* is the same as *def prep_score* but this function adjust the highscore.
- *def show* is to draw the score into the screen

```
7   class Scoreboard():
8       def __init__(self, ai_settings, screen, stats):
9           self.screen = screen
10          self.screen_rect = screen.get_rect()
11          self.stats = stats
12          self.ai_settings = ai_settings
13
14          #font settings for scoring
15          self.text_color = (250,250,250)
16          self.font = pygame.font.SysFont(None, 30)
17
18          self.prep_score()
19          self.prep_high_score()
20          self.prep_player()
21
22      #turn the score into a rendered image
23      def prep_score(self):
24          rounded_score = int(round(self.stats.score, -1))
25          score_str = "{:,}".format(rounded_score)
26          self.score_image = self.font.render(score_str, True, self.text_color)
27          self.score_rect = self.score_image.get_rect()
28          self.score_rect.right = self.screen_rect.right - 20
29          self.score_rect.top = 20
30          print(self.stats.score)
31
32      #draw the score to the screen
33      def show_score(self):
34          self.screen.blit(self.score_image, self.score_rect)
35          self.screen.blit(self.high_score_image, self.high_score_rect)
36          self.players.draw(self.screen)
37
```

Scoreboard › __init__()

```
29          self.score_rect.top = 20
30          print(self.stats.score)
31
32      #draw the score to the screen
33      def show_score(self):
34          self.screen.blit(self.score_image, self.score_rect)
35          self.screen.blit(self.high_score_image, self.high_score_rect)
36          self.players.draw(self.screen)
37
38      def prep_high_score(self):
39          high_score = int(round(self.stats.high_score, -1))
40          high_score_str = "{:,}".format(high_score)
41          self.high_score_image = self.font.render(high_score_str, True, self.text_color)
42
43          #Center the high score at the top of the screen
44          self.high_score_rect = self.high_score_image.get_rect()
45          self.high_score_rect.centerx = self.screen_rect.centerx
46          self.high_score_rect.top = self.score_rect.top
47
48      def prep_player(self):
49          self.players = Group()
50          for player_number in range(self.stats.ships_left):
51              player = Player(self.ai_settings, self.screen)
52              player.rect.x = -20
53              player.rect.y = -20
54              self.players.add(player)
55
56
57
58
59
```

Scoreboard › __init__()

Class Game Stats (*game_stats.py)*

- This class is to set the game statistics such as the player's life limit and to reset the stats when the game restarts

```
class GameStats():

    def __init__(self, ai_settings):
        self.ai_settings = ai_settings
        self.reset_stats()
        self.game_active = True
        self.high_score = 0

        self.game_active = False

    def reset_stats(self):
        self.ships_left = self.ai_settings.ship_limit
        self.score = 0
```

Functions (*functions.py*)

- *def check_events*() :

  The usage of this function is to manage events so it can respond to a keypresses and mouse events.

- *def play_button*():

  This function is to check if the play button whether it's inactive when the program first start.

- *def check_keydown_events*():

  This function is to check any key push input. I also input a sound effect on the spacebar key using *pygame.mixer*, so it will make a noise everytime the player shoot a laser.

- *def check_update_bullets*():

  This function is to update the bullets position and to run check_bullets_collision.

- *def check_bullet_collision*():

  This function is to create a collison events and its response and also it will update the score whenever the bullet hits an enemy.

- *def update_enemy*():

  This function is to update the enemy against collision with the player or against the left edge of the screen.

- *def ship_hit*():

  This function is to reset the player's position when the enemy hits the player and it will also update the stats.

- *def check_highscore*():

  This function is to check if there is a new highscore and it will update the highscore on the top of the screen.

- def check_enemy_left():

  This function is to check if the enemies have reached the left edge of the screen and if it's true, it will run *def ship_hit*.

- def update_screen():

  This function is to update all the events that is happening on the screen such as the player's movement, the bullets, spawning the enemy, etc.

```
import sys

import pygame,time

from bullet import Bullet
from pygame.sprite import *
from pygame import *
from enemy import Enemy
from time import sleep



def check_events(ai_settings,screen,stats,sb, play_button,  player,enemies,bullets):
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

        elif event.type == pygame.MOUSEBUTTONDOWN:
            mouse_x, mouse_y = pygame.mouse.get_pos()
            check_play_button(ai_settings, screen, stats,sb, play_button, player, enemies, bullets, mouse_x, mouse_y)


        elif event.type == pygame.KEYDOWN:
            check_keydown_events(event,ai_settings,screen,player,bullets)
            if event.key == pygame.K_DOWN:
                player.moving_down = True
            elif event.key == pygame.K_UP:
                player.moving_up = True
            elif event.key == pygame.K_LEFT:
                player.moving_left = True
            elif event.key == pygame.K_RIGHT:
```

check_enemies_l...

```
        elif event.type == pygame.KEYUP:
            if event.key == pygame.K_DOWN:
                player.moving_down = False
            if event.key == pygame.K_UP:
                player.moving_up = False
            elif event.key == pygame.K_LEFT:
                player.moving_left = False
            elif event.key == pygame.K_RIGHT:
                player.moving_right = False

#function to check the play button when the program first start
def check_play_button(ai_settings, screen, stats,sb, play_button,player, enemies,bullets, mouse_x,mouse_y):
    #to reset the game each time user click the play button
    button_clicked = play_button.rect.collidepoint(mouse_x, mouse_y)
    if button_clicked and not stats.game_active:
        stats.game_active = True
        stats.reset_stats()
        pygame.mouse.set_visible(False)
        enemies.empty()
        bullets.empty()
        sb.prep_score()
        sb.prep_high_score()
        sb.prep_player()



        player.center_player()

def check_keydown_events(event,ai_settings,screen,player,bullets):
    if event.key == pygame.K_SPACE:
        if len(bullets) < ai_settings.bullets_allowed:
```

check_enemies_l...

```

def check_keydown_events(event,ai_settings,screen,player,bullets):
    if event.key == pygame.K_SPACE:
        if len(bullets) < ai_settings.bullets_allowed:
            pygame.mixer.pre_init(44100, -16,2,2048)
            pygame.mixer_music.load('sound/Laser Blasts-SoundBible.com-108608437.mp3')
            pygame.mixer.music.set_volume(0.1)
            pygame.mixer.music.play(0)
            new_bullet = Bullet(ai_settings,screen,player)
            bullets.add(new_bullet)

#function to update the bullets position and the collisions
def update_bullets(ai_settings, screen,stats,sb, player, enemies, bullets):
    bullets.update()
    check_bullet_enemy_collisions(ai_settings,screen, stats, sb, player,enemies,bullets)

#to check if the bullet hit the enemy ship
def check_bullet_enemy_collisions(ai_settings, screen,stats, sb, player, enemies, bullets):
    collisions = pygame.sprite.groupcollide(bullets, enemies, True, True)
    for bullet in bullets.copy():
        if bullet.rect.bottom <= 0:
            bullets.remove(bullet)

    if collisions:
        stats.score += ai_settings.enemy_points
        sb.prep_score()
    check_high_score(stats, sb)


def update_enemy(ai_settings, screen, stats, sb,player, enemies, bullets):
    if pygame.sprite.spritecollideany(player, enemies):
```

check_enemies_l...

```
94
95      #to reset the player's position when collision with enemies
96      def ship_hit(ai_settings, screen,stats, sb,player, enemies, bullets):
97          if stats.ships_left > 0:
98              stats.ships_left -= 1
99              enemies.empty()
100             bullets.empty()
101             player.center_player()
102             sb.prep_player()
103
104             sleep(0.5)
105
106         else:
107             stats.game_active = False
108             pygame.mouse.set_visible(True)
109
110
111     def check_high_score(stats, sb):
112             """Check to see if there's a new highscore"""
113             if stats.score > stats.high_score:
114                 stats.high_score = stats.score
115                 sb.prep_high_score()
116
117     def check_enemies_left(ai_settings, screen, stats,sb, player,enemies, bullets):
118         """Check if enemies have reached the left screen"""
119         screen_rect = screen.get_rect()
120         for enemy in enemies.sprites():
121             if enemy.rect.left <= screen_rect.left:
122                 ship_hit(ai_settings, screen,stats,sb, player, enemies, bullets)
123                 break
124
```

check_enemies_l...

```
122                 ship_hit(ai_settings, screen,stats,sb, player, enemies, bullets)
123                 break
124
125     #function to update the screen
126     def update_screen(ai_settings, screen, stats, sb, player,enemies,bullets, play_button):
127         ai_settings.levelup(stats.score)
128
129         for bullet in bullets.sprites():
130             bullet.draw_bullet()
131         #to spawn random enemies
132         if pygame.time.get_ticks()%80 == 0:
133             new_enemy = Enemy(ai_settings, screen)
134             enemies.add(new_enemy)
135         if not stats.game_active:
136             play_button.draw_button()
137         sb.show_score()
138         player.blitme()
139
140         if stats.game_active:
141             for en in enemies:
142                 en.update()
143                 en.blitme()
144
145
146         pygame.display.flip()
147
148
149
150
```

check_enemies_l...

Main (*Main.py*) **:**

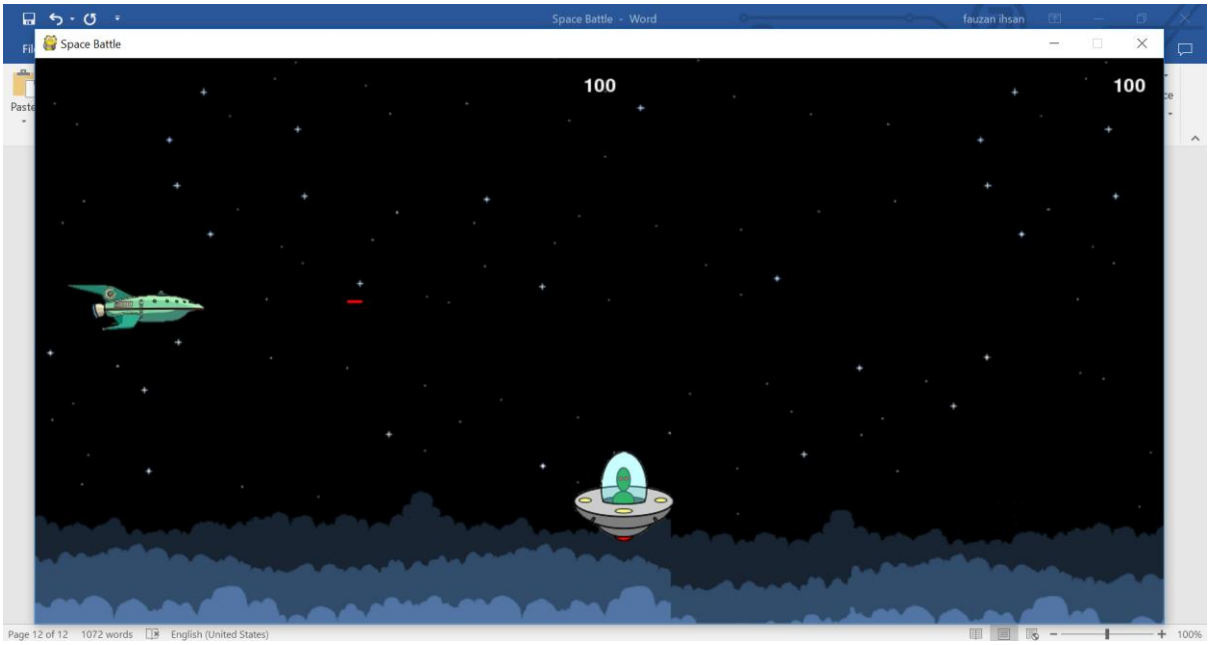
- *def main*():

  This is the main function that runs and contains all of the other functions. Inside main function, there is the main loop that contains updates such as the *check_events*, *update_screen*, *update_bullets*, *update_enemy*, background and other event loop and codes that manages the screen.

```python
import pygame
import sys


from settings import Settings
from player import Player
from pygame.sprite import Group
from enemy import Enemy
from game_stats import GameStats
from scoreboard import Scoreboard
from button import Button

import functions as gf

#the main function
def main():
    pygame.init()
    ai_settings = Settings()
    screen = pygame.display.set_mode((ai_settings.screen_width, ai_settings.screen_height))

    bullets = Group()
    enemies = Group()
    enemybullets = Group()
    enemy = Enemy(ai_settings, screen)

    stats = GameStats(ai_settings)
    sb = Scoreboard(ai_settings, screen, stats)

    play_button = Button(ai_settings, screen, "Play")


```

```python
    player = Player(ai_settings, screen)
    bground = pygame.image.load('images/wallhaven-41034.bmp')
    x = 0


    #main loop
    while True:
        gf.check_events(ai_settings, screen, stats,sb, play_button, player,enemies, bullets)

        if stats.game_active:
            player.update()
            gf.update_bullets(ai_settings, screen, stats, sb, player, enemies,bullets)
            gf.update_enemy(ai_settings, screen, stats,sb, player,enemies, bullets)

        gf.update_screen(ai_settings, screen, stats, sb, player, enemies, bullets, play_button)

        #moving background
        screen.fill((0,0,0))
        rel_x = x % bground.get_rect().width
        screen.blit(bground, (rel_x - bground.get_rect().width,0))
        if rel_x < ai_settings.screen_width:
            screen.blit(bground, (rel_x,0))

        x -= 3


main()

#Special thanks to longlong,arkaan,and python crash course for the help of making my final project#

```

IV. Evidence Of Working



V. References

1. Python Crash Course
2. www.youtube.com
3. www.stackoverflow.com