上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

# 《Introduction to business computing》

## Class Project Report

Our Project: Here Come Aliens!

Our Team Name: X-Girls

Faculty advisor: Bao Yang

Team Members: 朱芙蓉 517120910186

刘　颖 517120910175

周韩韵 517120910185

Wendy Zhou 715120290004

# Here Come Aliens

## Table of Contents

- Create  aliens

**Run our game**

**Weakness**

**Conclusion**

# Alien Invasion

## Abstract

For our Python project, we chose to contribute to the online gaming industry by creating and coding a game called “Alien Invasion.” Through this game, we are solving the need of diversity with online games and helping to provide fun and entertainment to users of all ages. The game was coded with Python and, specifically, the PYGAME module, through which we were able to implement the interactive nature of the game and apply various computer graphics and design elements.

“Alien Invasion” not only has real-world applications but also allows us to expand our Python knowledge by building a completely operational and interactive video game, adding fun to users’ everyday lives.

## Reason for Choosing This Topic

The online gaming industry has been booming in recent years, with the data volume of global online gaming expected to grow to 568 petabytes in 2020 and the worldwide market projected to reach 2.2 trillion USD by 2021. Not only is the industry very lucrative, but it has endless possibilities for enhancement and innovation through different programming and design techniques. The evolution of the gaming industry just within the past few years is remarkable, and speaks volumes about the industry's potential to grow and change the way that society consumes digital entertainment.

As part of a younger millennial generation in an increasingly technology-focused world, learning the back-end behind how video games are made was very appealing to us as a group. China as a society is a large consumer of video games, so we felt it was something that fit well with the current Chinese youth culture, as well.

Our "Alien Invasion" game is a new take on a classic video game style. In "Alien Invasion," one person plays at a time and tries to keep their spacecraft alive while defending it from aliens. The game is simple and easy-to-learn, but also engaging with various levels of difficulty.

## The tools we use

### 1. PYGAME

PYGAME is a cross platform Python module, specially designed for video games, including images and sounds. Based on SDL, it allows real-time video games to be developed without being fettered by low-level languages, such as machine language and assembly language.

It contains images, sounds. Based on SDL, it allows real-time video games to be developed without being fettered by low-level languages, such as machine language and assembly language. Based on this idea, all the required game functions and ideas (mainly image aspects) are completely simplified to the game logic itself, and all resource structures can be provided by a high-level language, such as Python.

### 2. PHOTOSHOP

Adobe Photoshop, referred to as "PS", is an image processing software developed and distributed by Adobe Systems.

Photoshop mainly deals with digital images made up of pixels. With its numerous editing and drawing tools, it can effectively edit pictures. PS has many functions, such as image, graphics, text, video, publishing and so on.

## The process of coding

### Ⅰ. Install PYGAME and create a spaceship that can move right and left

### (1) Install PYGAME

For windows 10, python 3.6:

```
$ pip install wheel
```

```
$ pip install pygame-1.9.3-cp36-cp36m-win_amd64.whl
```

## (2) Create a Pygame window and respond to user input

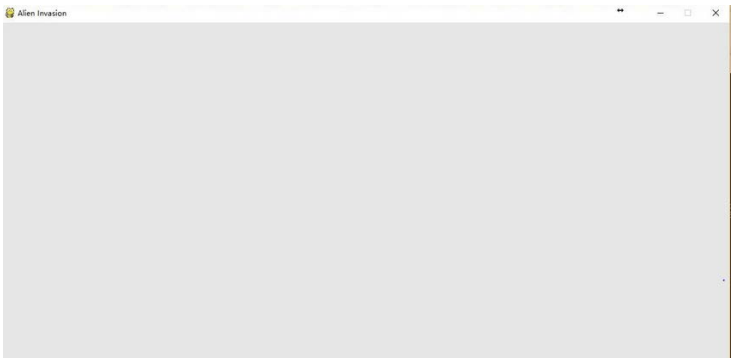
① Create a folder alien_invasion and create a new alien_invasion.py file in the folder.

Enter the following code.

```
import sysimport pygamedef run_game():

    #initialize game and create a dispaly object

    pygame.init()

    screen = pygame.display.set_mode((1200,800))

    pygame.display.set_caption("Alien Invasion")

    # set backgroud color

    bg_color = (230,230,230)



    # game loop

    while True:

        # supervise keyboard and mouse item

        for event in pygame.event.get():

            if event.type == pygame.QUIT:
```

```
                sys.exit()

        # fill color

        screen.fill(bg_color)

        # visualiaze the window

        pygame.display.flip()

run_game()
```

② By running the above code, we can get a grey interface window:




## (3) Create a set class

① In order to create some new features easily in the process of writing games, an additional settings module is written, which contains a Settings class that stores all the settings in one place. This makes it easier to modify the appearance of the game when the project becomes larger.

We first modify the size of the display screen and the color of the display screen in alien_invasion.py.

First, create a new Python file settings.py under the alien_invasion folder and add the following code to it:

```
class Settings(object):

    """docstring for Settings"""

    def __init__(self):

        # initialize setting of game



        # screen setting

        self.screen_width = 1200

        self.screen_height = 800

        self.bg_color = (230,230,230)
```

②Then import the Settings class into alien_invasion.py and use the relevant settings to revise the following:

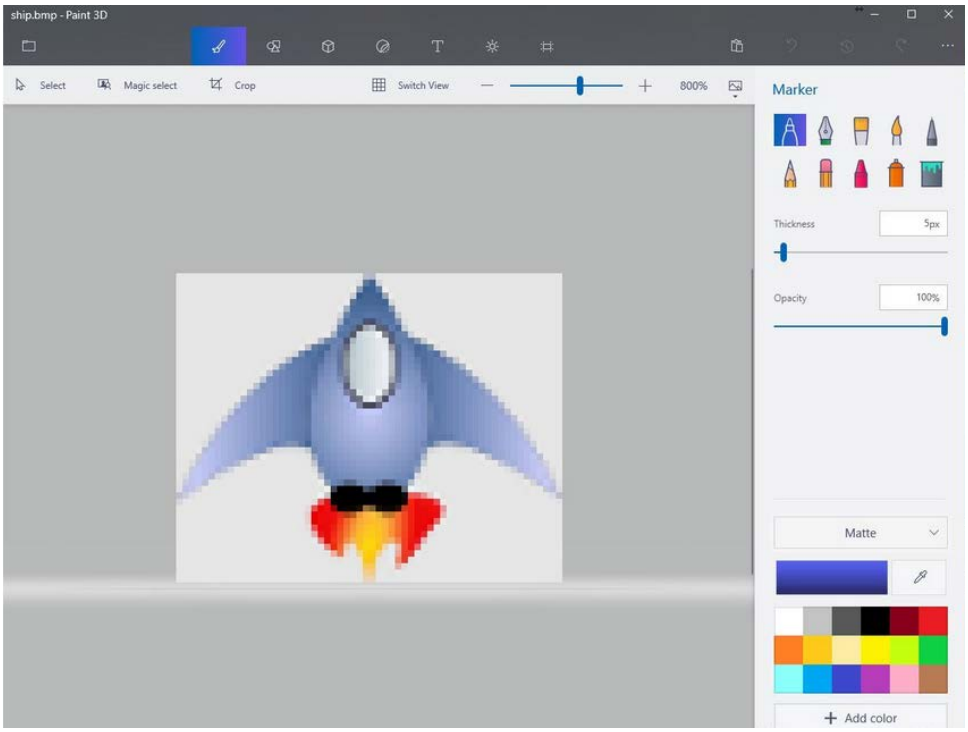
```
      import sysimport pygamefrom settings import Settingsdef run_game():

    #initialize game and create a dispaly object

    pygame.init()
```

```
    ai_settings = Settings()

    screen = pygame.display.set_mode((ai_settings.screen_width,ai_settings.
screen_height))

    pygame.display.set_caption("Alien Invasion")

    # set backgroud color

    bg_color = (230,230,230)



    # game loop

    while True:

        # supervise keyboard and mouse item

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                sys.exit()

        # fill color

        screen.fill(ai_settings.bg_color)

        # visualiaze the window

        pygame.display.flip()

run_game()
```

## （4）Adding a spacecraft image

①Next, we need to add the spaceship to the game. To draw the player's ship on the screen, we will load an image and use Pygame  Blit() method to draw it.

Almost all kinds of image files can be used in games, but using bitmap (.Bmp) files is the most simple, because Pygame loads bitmaps by default. Although other types of images can also be loaded, additional libraries are required. We create a new folder called images in the main project folder (alien_invasion), and put the following BMP picture into it.




②Next, we create the spaceship class ship.py:

```
import pygameclass Ship():
```

```
def __init__(self,screen):
    #initialize spaceship and its location
    self.screen = screen

    # load bmp image and get rectangle
    self.image = pygame.image.load('image/ship.bmp')
    self.rect = self.image.get_rect()
    self.screen_rect = screen.get_rect()

    #put spaceship on the bottom of window
    self.rect.centerx = self.screen_rect.centerx
    self.rect.bottom = self.screen_rect.bottom

def blitme(self):
    #buld the spaceship at the specific location
    self.screen.blit(self.image,self.rect)
```

③Finally, we draw the spacecraft on the screen, that is, calling the blitme method in the alien_invasion.py file.

```
import sysimport pygamefrom settings import Settingsfrom ship import Setti
ngsdef run_game():

    #initialize game and create a dispaly object

    pygame.init()

    ai_settings = Settings()

    screen = pygame.display.set_mode((ai_settings.screen_width,ai_settings.
screen_height))

    ship = Ship(screen)

    pygame.display.set_caption("Alien Invasion")

    # set backgroud color

    bg_color = (230,230,230)



    # game loop

    while True:

        # supervise keyboard and mouse item

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                sys.exit()

        # fill color

        screen.fill(ai_settings.bg_color)
```

```
        ship.blitme()

        # visualiaze the window

        pygame.display.flip()

run_game()
```

# II.Refactoring: module game_functions

In large projects, it is often necessary to reconstruct existing code before adding new code. The purpose of refactoring is to simplify the structure of the code and make it easier to expand. We will implement a game_functions module that will store a large number of functions that enable the game Alien invasion to run. By creating the module game_functions, alien_invasion.py can be avoided too long to make its logic easier to understand.

## (1)Function check_events ()

①First, we move the code to manage the event into a function called check_events (), in order to isolate the event loop.

```
import sysimport pygame

def check_events():

    #respond to  keyboard and mouse item

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            sys.exit()
```

②Then we modify the alien_invasion.py code, import the game_functions module, and replace the event loop with the call to the pair function check_events ().

```
import sysimport pygamefrom settings import Settingsfrom ship import Shipi
mport game_functions as gfdef run_game():

    #initialize game and create a dispaly object

    pygame.init()

    ai_settings = Settings()

    screen = pygame.display.set_mode((ai_settings.screen_width,ai_settings.
screen_height))

    ship = Ship(screen)

    pygame.display.set_caption("Alien Invasion")

    # set backgroud color

    bg_color = (230,230,230)



    # game loop

    while True:

        # supervise keyboard and mouse item

        gf.check_events()

        # fill color
```

```
        screen.fill(ai_settings.bg_color)

        ship.blitme()

        # visualiaze the window

        pygame.display.flip()

run_game()
```

Using the same way, move the code of the update screen to a update_screen () function and place the function in module game_functions.

# III. Drive the ship

## (1) Response key

After we detect the KEYDOWN event through event.type, we need to further determine which key is it. The code is as follows:

```
def check_events(ship):

    #respond to  keyboard and mouse item

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            sys.exit()

        elif event.type == pygame.KEYDOWN:
```

```
            if event.key == pygame.K_RIGHT:
                #move right
                ship.rect.centerx +=1
```

## (2) Allow the ship to move continuously

We use the ship class to control it, so we add an attribute name called, moving_right, and a update () method to detect the status of the flag moving_right.

```
        self.moving_right = False
    def update(self):
        if self.moving_right:
            self.rect.centerx +=1

        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RIGHT:
                #move right
                ship.moving_right = True
        elif event.type == pygame.KEYUP:
            if event.key = pygame.K_RIGHT:
                ship.moving_right = False
```

```
    while True:

        # supervise keyboard and mouse item

        gf.check_events(ship)

        ship.update()
```

### （3）Adjust the speed of the ship

At present, each time the while loop is executed, the spacecraft moves up to one pixel at most, and we can add ship_speed_factor to the Settings to control the speed of the spacecraft. We will determine how much distance the spacecraft will move at most times in each cycle.

```
class Settings(object):

    """docstring for Settings"""

    def __init__(self):

        # initialize setting of game



        # screen setting

        self.screen_width = 1200

        self.screen_height = 800

        self.bg_color = (230,230,230)

        self.ship_speed_factor = 1.5
```

```
class Ship():



    def __init__(self,ai_settings,screen):

        #initialize spaceship and its location

        self.screen = screen

        self.ai_settings = ai_settings
```

## （4）Reconstruction

Here we mainly talk about the refactoring of the check_events () function, which divide part of the code into two parts, one part of the KEYDOWN event, and a part of the KEYUP event.

```
def check_keydown_events(event,ship):

    if event.key == pygame.K_RIGHT:

        #move right

        ship.moving_right = True

    elif event.key == pygame.K_LEFT:

        #move right

        ship.moving_left = True def check_keyup_events(event,ship):

    if event.key == pygame.K_RIGHT:
```

```
        ship.moving_right = False

    elif event.key == pygame.K_LEFT:

        #move right

        ship.moving_left = Falsedef check_events(ship):

    #respond to  keyboard and mouse item

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            sys.exit()

        elif event.type == pygame.KEYDOWN:

            check_keydown_events(event,ship)

        elif event.type == pygame.KEYUP:

            check_keyup_events(event,ship)
```

# IV. Shoot

## (1) Add bullets

① Create the Bullet class

Here we create a dark grey bullet with a width of 3 pixels and a height of 15 pixels. The speed of the bullet is a little lower than that of the ship.

```
import pygamefrom pygame.sprite import Sprite

class Bullet(Sprite):

    """A class to manage bullets fired from the ship."""



    def __init__(self, ai_settings, screen, ship):

        """Create a bullet object, at the ship's current position."""

        super().__init__()

        self.screen = screen



        # Create bullet rect at (0, 0), then set correct position.

        self.rect = pygame.Rect(0, 0, ai_settings.bullet_width,

            ai_settings.bullet_height)

        self.rect.centerx = ship.rect.centerx

        self.rect.top = ship.rect.top



        # Store a decimal value for the bullet's position.

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



        self.color = ai_settings.bullet_color
```

```
        self.speed_factor = ai_settings.bullet_speed_factor


    def update(self):
        """Move the bullet up the screen."""
        # Update the decimal position of the bullet.
        self.y -= self.speed_factor
        # Update the rect position.
        self.rect.y = self.y


    def draw_bullet(self):
        """Draw the bullet to the screen."""
        pygame.draw.rect(self.screen, self.color, self.rect)
```

② Store the bullet in group

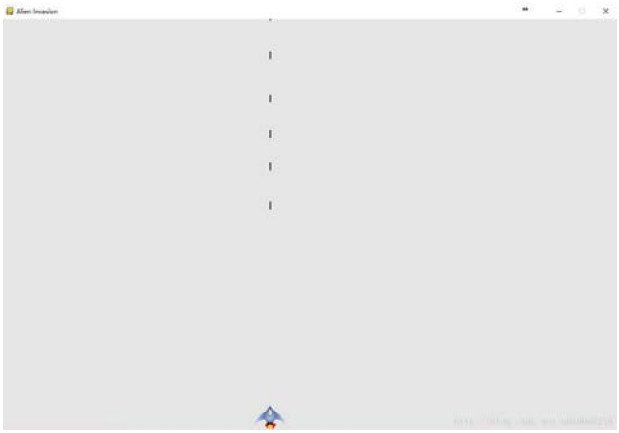
First, we create a group in alien_invasion to store all effective bullets.

```
def run_game():
    #initialize game and create a dispaly object
    pygame.init()
    ai_settings = Settings()
```

```
    screen = pygame.display.set_mode((ai_settings.screen_width,ai_settings.
screen_height))

    ship = Ship(ai_settings,screen)

    bullets = Group()

    pygame.display.set_caption("Alien Invasion")

    # set backgroud color

    bg_color = (230,230,230)



    # game loop

    while True:

        # supervise keyboard and mouse item

        gf.check_events(ai_settings, screen, ship,bullets)

        ship.update()

        bullets.update()

        gf.update_screen(ai_settings, screen, ship,bullets)
```

③. Fire

We modify the check_keydown_events() function to listen for events when a player presses a space key.We also modify the update_screen() function to ensure that each bullet is redrawn every time the screen is updated.



④ Delete the missing bullet

```
import sysimport pygamefrom settings import Settingsfrom ship import Shipi
mport game_functions as gffrom pygame.sprite import Groupdef run_game():

    #initialize game and create a dispaly object

    pygame.init()

    ai_settings = Settings()

    screen = pygame.display.set_mode((ai_settings.screen_width,ai_settings.
screen_height))

    ship = Ship(ai_settings,screen)

    bullets = Group()

    pygame.display.set_caption("Alien Invasion")

    # set backgroud color

    bg_color = (230,230,230)
```

```
    # game loop

    while True:

        # supervise keyboard and mouse item

        gf.check_events(ai_settings, screen, ship,bullets)

        ship.update()

        bullets.update()

        for bullet in bullets.copy():

            if bullet.rect.bottom <=0:

                bullets.remove(bullet)

        gf.update_screen(ai_settings, screen,ship,bullets)

run_game()
```

## ⑤Limit the number of bullets

We have a rule that only three bullets can exist on the screen at the same time. We o nly need to check whether the number of bullets that remain on the screen is less than thr ee before each bullet is created.

```
def fire_bullet(ai_settings, screen, ship, bullets):

        """Fire a bullet, if limit not reached yet."""

       # Create a new bullet, add to bullets group.

        if len(bullets) < ai_settings.bullets_allowed:

        new_bullet = Bullet(ai_settings, screen, ship)

         bullets.add(new_bullet)
```

## V.Create aliens

```
class Alien(Sprite):

    """A class to represent a single alien in the fleet."""



    def __init__(self, ai_settings, screen):

        """Initialize the alien, and set its starting position."""

        super().__init__()

        self.screen = screen

        self.ai_settings = ai_settings



        # Load the alien image, and set its rect attribute.

        self.image = pygame.image.load('images/alien.bmp')

        self.rect = self.image.get_rect()



        # Start each new alien near the top left of the screen.

        self.rect.x = self.rect.width

        self.rect.y = self.rect.height
```

```
        # Store the alien's exact position.
        self.x = float(self.rect.x)

    def blitme(self):
        """Draw the alien at its current location."""
        self.screen.blit(self.image, self.rect)
```

In this way, we can create a group of aliens.

# Run our game

## Ⅰ.Move and shoot the aliens

We set the alien's speed in the Settings class and then use the update method in the Alien class to implement the move.

Detect the collision immediately after we have updated the position of the bullet.

## Ⅱ.When to end the game

（1） Here we also need to know when to end the game.

- All the ships were destroyed
- The aliens arrive at the bottom of the screen

（2）Here is an attribute game_active added to GameStats to mark the end of the game when the player's spacecraft is finished.

```
def __init__(self, settings):
        --snip-
        # 游戏刚启动时处于活动状态
          self.game_active = True

def ship_hit(ai_settings, stats, screen, ship, aliens, bullets):
        """The spacecraft was hit by an alien. """
        if stats.ships_left > 0:
```

```
            stats.ships_left -= 1
            --snip-
            sleep(0.5)
        else:
            stats.game_active = False
```

## Weaknesses

1) Lack of complexity

Nowadays, video games have become quite complex and realistic, involving multiplayers and extremely detailed graphics. For these games, the back-end development requires much heavier programming than can be covered in Python or Pygame. In comparison, “Alien Invasion” is much simpler and basic when it games to game rules and design.

One way in which “Alien Invasion” could have been improved is using more high-quality images and graphics, or making the game able to be played simultaneously by two players.

2) Pygame

While Pygame is suitable and very easy-to-use for simple games such as “Alien Invasion”, it is not good for handling games requiring high performance or resolution. Since Pygame operates through the CPU, it is also slower to handle graphics.

## Conclusion

It was a unique experience being able to experiment with game development through our application of Python and Pygame. We were able to strengthen our Python and general programming skills while making something that was functional and operational in the real world. By expanding on topics touched upon in class, we also were able to broaden our design skills while creating the interface of the game, which was a great learning moment. Overall, the project taught us a lot, and we are very satisfied with our end product.