# BattleShips

By Yimeng Zhang, Jialin Wang and Jiaxing Lu

# Project description

We built a python version of the classic board game Battleships, which is played on labeled grids on which each player's fleet of ships are marked. The locations of the fleets are concealed from the other player. Players alternate turns calling "shots" at the other player's ships, and the objective of the game is to destroy the opposing player's fleet.

We implemented the “Chain fire “game mode, where the number of shots a player fires is equal to the number of ships the player has remaining, and two game modes: Player vs Player and Player vs Computer

# Code Framework

## Two Classes

1. class Ship
2. class Battlefield

# class Ship()---Attributes

```
class Ship:
    def __init__(self, length, direction):
        self.length = length
        self.direction = direction
        self.actual_location = []
```

- Initiate basic attributes of a ship

# class Ship()---Method

```
def set_ship(self, locationX, locationY):
    if self.direction == 'V': #verticle
        for i in range(self.length):
            the_boat = [locationX, locationY + i]
            self.actual_location.append(the_boat)
    elif self.direction == 'H':  # horizontal
        for i in range(self.length):
            the_boat = [locationX + i, locationY]
            self.actual_location.append(the_boat)
```

- The player only inputs a point, the orientation and length of the ship, we need to record the actual location (more than one point) in a list

# Class Battlefield()---Attributes & Basic Methods

```
class Battlefield:
    def __init__(self, width=8, length=8):
        self.battlefield = [['o ' for i in range(length + 1)] for j in range(width + 1)]
        for i in range(length + 1):
            self.battlefield[0][i] = str(i)+" "
        for j in range(width + 1):
            self.battlefield[j][0] = str(j)+" "
        self.width = width
        self.length = length
    def init_my_battlefield(self, ship):
        for actual_boat in ship.actual_location:
            self.battlefield[actual_boat[1]][actual_boat[0]] = '@ '
    def __str__(self):
        str=""
        for i in range(self.width+1):
            for j in range(self.length+1):
                str+=self.battlefield[i][j]
            str+="\n"
        return str
```

- Initiate the battlefield
- Override python print() function so that the battlefield can be easily printed out

# class Battlefield()---Method

```
l_length=[]
l_width=[]

d_roster={'Battleship':[5],'Cruiser':[4],'Destroyer':[3],'Corvette':[2]}


for i in range(1, b.length+1):
    l_length.append(i)
for i in range(1, b.width+1):
    l_width.append(i)
length = 5
for i in range(4):
    direction = input('Please enter the orientation of your ' + roster[i]+'\n')
    while direction != 'V' and direction != 'H':
        print('Please enter a valid orientation for your ship')
        direction = input('Please enter the orientation of your ' + roster[i]+'\n')
    try:
        x = int(input("Please set up the x coordinate of your " + roster[i]+'\n'))
        y = int(input("Please set up the y coordinate of your " + roster[i]+'\n'))
    except ValueError:
        continue
    while x not in l_length or (direction == "H" and x + length-1 > b.length) \
            or y not in l_width or (direction=="V" and y+length-1> b.width):
        print('Please enter a valid orientation and x/y coordinate to make sure it is not out of scope')
        direction = input('Please enter the orientation of your ' + roster[i])
        try:
            x = int(input("Please set up the x coordinate of your " + roster[i]+'\n'))
            y = int(input("Please set up the y coordinate of your " + roster[i]+'\n'))
        except ValueError:
            continue
    while b.battlefield[y][x] == "@ ":
        print("Please enter a valid orientation and x/y coordinate to make sure it is not the same as one of the previous boats")
        try:
                direction = input('Please enter the orientation of your ' + roster[i]+'\n')
                x = int(input("Please set up the x coordinate of your " + roster[i]+'\n'))
                y = int(input("Please set up the y coordinate of your " + roster[i]+'\n'))
        except ValueError:
                continue
    a_ship = Ship(length, direction, 1)
    a_ship.set_ship(x, y)
    for j in a_ship.actual_location:
        d_roster[roster[i]].append(j)
    b.init_my_battlefield(a_ship)
    os.system('clear') #clears previous output so that the player sees what appears to be a rapid update of the
                       #battlefield without repetition
    print(b)
    length -= 1
return b, d_roster # d_roster is used to check whether a ship is sunk
```

User prompt: input setup Taking IndexError and ValueError into consideration

To record the actual location, battlefield, and the condition of a ship

# Game Stage 1: Set up

```
def main():
    print("**********Welcome, Player 1! The game has started! Enjoy and try to win!**********\n")
    b1, d_player1_roster = set_battlefield()
    a=input('Press enter to continue.\n')
    os.system('clear')
    print("**********Welcome, Player 2! The game has started! Enjoy and try to win!**********\n")
    b2, d_player2_roster= set_battlefield()
    a=input('Press enter to continue.\n')
    b1_for_player2 = Battlefield()
    b2_for_player1 = Battlefield()
    player1_attacks, player2_attacks=2,2
    l_player1_destroyed=[]
    l_player2_destroyed=[]
```

- Both players start by placing their fleet onto their own battlefield
- Two blank battlefields are created to be displayed as player 1's battlefield to player 2, and vice versa
- Two lists are created to hold the destroyed ships of player 1 and two

# Game Stage 2: Attack

```
def attack(actual_battlefield,displayed_battlefield):
    l_length=[]
    l_width=[]
    for i in range(1, actual_battlefield.length+1):
        l_length.append(i)
    for i in range(1, actual_battlefield.width+1):
        l_width.append(i)

    x = int(input("Guess one x coordinate of your opponent's ship "))
    y = int(input("Guess one y coordinate of your opponent's ship "))
    while x not in l_length or y not in l_width:
        print('Please enter a valid x/y coordinate to make sure it is in the enemy field')
        x = int(input("Guess one x coordinate of your opponent's ship "))
        y = int(input("Guess one y coordinate of your opponent's ship "))



    actual_battlefield.update_battlefield(x,y,displayed_battlefield)
```

- Attack function takes player input coordinates and compare them with the opponent's battlefield, updating their displayed battlefield and printing it out onto the player's screen

# Game Stage 2: Attack

```
def update_battlefield(self, Xcoordinate, Ycoordinate,battlefield):

    if battlefield.check_coordinate(Xcoordinate,Ycoordinate)==0:
        print("\nThis point has already been hit, try another one next time!")
        attack(self, battlefield)
        return
    elif self.check_coordinate(Xcoordinate,Ycoordinate)==1:
        battlefield.battlefield[Ycoordinate][Xcoordinate] = '  '
        self.battlefield[Ycoordinate][Xcoordinate] = '  '
        print("\nSorry, you didn't hit an enemy ship!")
    elif self.check_coordinate(Xcoordinate, Ycoordinate)==2:
        battlefield.battlefield[Ycoordinate][Xcoordinate] = 'x '
        self.battlefield[Ycoordinate][Xcoordinate] = 'x '
        print("\nGood job, you hit an enemy ship!")

    print("Enemy battlefield:")
    print(battlefield)
     #used to grant a player a second chance if player chooses to fire on a tile that has already been hit
```

If the player selects a previously hit tile, he will receive another chance

- We compare the player's input x,y with his opponent's battlefield, if it is the same as the opponent's ship location, update the player's own battlefield with "x", otherwise the tile will turn into " "

# Game Stage 3: Players take turns to attack

```
for i in range(5):
    shots_player1=2-len(l_player1_destroyed)
    shots_player2=2-len(l_player2_destroyed)

    os.system('clear')
    print("Player1's turn:\n")
    print('Enemy battlefield:')
    print(b2_for_player1)#displays the current state of the opponent's battlefield for player to consult while planning attacks
    print('Your battlefield:')
    print(b1)

    print('Casualties:',l_player1_destroyed)
    print('Kills:',l_player2_destroyed)
    print('Number of shots: %s'%shots_player1)

    for i in range(player1_attacks):
        print('shot number %s' %(i+1))

        attack(b2,b2_for_player1)
        for k, v in d_player2_roster.items():
            HP=v[0]
            for j in v:
                if type(j)==list:
                    if b2.battlefield[j[1]][j[0]]=='x ':
                        HP-=1
            if HP==0:
                player2_attacks-=1
                l_player2_destroyed.append(k)

        for i in l_player2_destroyed:
            d_player2_roster.pop(i,None)

        if len(l_player2_destroyed)==2:
            print ('Player 1 has won through eliminating Player 2\'s fleet.')
            return
    a=input('Press enter to finish your turn') #allows the player to have a glimpse of the battlefield after their final shot
```

```
for i in range(5):
    shots_player1=2-len(l_player1_destroyed)
    shots_player2=2-len(l_player2_destroyed)
```

Shot number is determined by the condition of player's own ships (whether sunk or not)
(in this demo version a player only has 2 ships, so max=2 shots)

# Game Stage 3: Players take turns to attack

```
os.system('clear')
print("Player1's turn:\n")
print('Enemy battlefield:')
print(b2_for_player1)#displays the current state of the opponent's battlefield for player to consult while planning attacks
print('Your battlefield:')
print(b1)

print('Casualties:',l_player1_destroyed)
print('Kills:',l_player2_destroyed)
print('Number of shots: %s'%shots_player1)
```

Current state of the game displayed to each player at the beginning of their turn

# Game Stage 3: Players take turns to attack

```
for i in range(player1_attacks):
    print('shot number %s' %(i+1))

    attack(b2,b2_for_player1)
    for k, v in d_player2_roster.items():
        HP=v[0]
        for j in v:
            if type(j)==list:
                if b2.battlefield[j[1]][j[0]]=='x ':
                    HP-=1
        if HP==0:
            player2_attacks-=1
            l_player2_destroyed.append(k)

    for i in l_player2_destroyed:
        d_player2_roster.pop(i,None)

    if len(l_player2_destroyed)==2:
        print ('Player 1 has won through eliminating Player 2\'s fleet.')
        return
```

```
if len(l_player2_destroyed)==2:
    print ('Player 1 has won through eliminating Player 2\'s fleet.')
    return
```

Monitors fleet state after every shot and ends game if the fleet of one player is completely destroyed

# Game Stage 4: Calculate the points and announce the winner

```
def points_calculator(self):
    count = 0
    for i in range(1, self.length + 1):
        for j in range(1, self.width + 1):
            if self.battlefield[j][i] == 'x ':
                count += 1
    return count
```

```
p1=b1.points_calculator()
p2=b2.points_calculator()
if p1>p2:
    print("Player 1 wins!")
elif p1<p2:
    print("Player 2 wins!")
else:
    print("You two got the same score! Slug it out next time!")
```

For Player vs. Computer, same logic, but we generate random setup and attack instead.

```
for i in range(4):
    k = random.randint(0, 1)
    list0 = ['H', 'V']
    direction = list0[k]
    x = random.randint(5, 8)
    y = random.randint(1, 8)
    while (direction == "H" and x + length - 1 > 8) or (direction == "V" and y + length - 1 > 8):
        m = random.randint(0, 1)
        list1 = ['H', 'V']
        direction = list1[m]
        x = random.randint(5, 8)
        y = random.randint(1, 8)
    while b2.battlefield[y][x] == "@ ":
        n = random.randint(0, 1)
        list2 = ['H', 'V']
        direction = list2[n]
        x = random.randint(5, 8)
        y = random.randint(1, 8)
    a_ship = Ship(length, direction, 2)
    a_ship.set_ship(x, y)
    b2.init_my_battlefield(a_ship)
    length -= 1
return b2
```

# Implement into Chat System

```
if self.state == S_LOGGEDIN:
    # todo: can't deal with multiple lines yet
    if len(my_msg) > 0:
    ......
    elif my_msg[0:5]=='game1':
        self.out_msg += ' Have fun!\n\n'
        self.out_msg += '-----------------------------------\n'
        os.system('python game_single_player.py')
    elif my_msg[0:5]=='game2':
        self.out_msg += ' Have fun!\n\n'
        self.out_msg += '-----------------------------------\n'
        os.system('python game_two_players.py')
```

“game 1” for player vs. computer
“game 2” for player vs. player

# Online Double Game

```
if msg["action"]=="game":
    p=Battlefield(4,4,[])
    new=p.update_battlefield()
    msg["message"] += 'There are four gold sites to search among the sixteen sites\n'
    msg["message"] += 'Please enter the coordinate in order to find the gold\n'


else:
    try:
        clicker = msg["message"].split(',')
        for i in range(len(clicker)):
            clicker[i] = int(clicker[i])
        new.attack(clicker)
        if new.attack(clicker)==False:
            msg["message"] += "Continue and next one!\n"
            new.attack(clicker)
            if new.attack(clicker)==True:
                msg["message"] += "You are the champion!\n"
                msg["message"] += "Game over"
                msg["message"] +='This is the true map!'
                msg["message"] += 'x is gold site and o is empty!'
                for each in p.battlefield:
                    msg["message"] += each

        else:
            msg["message"] += "Game over"

    except:
        msg["message"] += " "
```

To be Continued...

# Demo

## Double_Players Mode

# GUI---Calculator

```
gui.title("A Small Calculator")

gui.geometry("300x150")

equation = StringVar()

expression_field = Entry(gui, textvariable=equation)

expression_field.grid(columnspan=4, ipadx=70)

equation.set('please enter your expression:')

button1 = Button(gui, text=' 1 ', fg='black', bg='red',
                 command=lambda: press(1), height=1, width=7)
button1.grid(row=2, column=0)

button2 = Button(gui, text=' 2 ', fg='black', bg='red',
                 command=lambda: press(2), height=1, width=7)
button2.grid(row=2, column=1)
```

```
plus = Button(gui, text=' + ', fg='black', bg='red',
              command=lambda: press("+"), height=1, width=7)
plus.grid(row=2, column=3)

minus = Button(gui, text=' - ', fg='black', bg='red',
               command=lambda: press("-"), height=1, width=7)
minus.grid(row=3, column=3)

multiply = Button(gui, text=' * ', fg='black', bg='red',
                  command=lambda: press("*"), height=1, width=7)
multiply.grid(row=4, column=3)

divide = Button(gui, text=' / ', fg='black', bg='red',
                command=lambda: press("/"), height=1, width=7)
divide.grid(row=5, column=3)

equal = Button(gui, text=' = ', fg='black', bg='red',
               command=equal_press, height=1, width=7)
```

# Problems and Challenges

1. The coordinates that players enter need to be checked to ensure that a: they remain within the scope of the battlefield, and b: that they do not .

   Solution: try-except to monitor the range of the coordinates, and check_coordinate method to decide whether a tile has been hit previously

2. Displaying the enemy battlefield after each shot is carried out by the player

   Solution: Setting up 4 battlefields in total, 2 for each player, one of which is the battlefield where the player places their vessels and the other being the battlefield that is displayed to their opponents

3. Making the computer in single player mode able to choose proper coordinates based on previous steps (choose coordinates near last success).

   Solution:  Add random number onto the success trial.

# Future Improvements

Add unique characteristics to different ship classes (e.g. Battleships have more guns and have a shield that can absorb extra shots, Destroyers can reveal a 2x2 area of the enemy battlefield, etc).

Implement an interface using pygame.