

## **-Anforderungsanalyse mit Komplexitätsschätzung**

User Story:

Als Spieler möchte ich alle feindlichen Schiffe treffen,  
bevor der Gegner alle meine Schiffe getroffen hat.

Moskau priorisierung:

Must have:

- Datenbank zur Speicherung der Schiffe
- Abbildung der Schiffe

Should have:

- Interface zur Eingabe der Koordinaten
- Working enemy AI

Could have:

- Optimierterer Code

Wont have:

- Schiffe mit mehr als einem Leben

## **-Planung**

Welches FramerWork und Warum:

**Web2py wurde gewählt zwecks der Benutzerfreundlichkeit des Interfaces**

Welche Views sind notwendig:

**Es wird nur eine View benötigt auf der eingabe der Daten und Abbildung des Spielfeldes möglich ist.**

Welche anderen Werkzeuge wurden verwendet:

```
import random  
from random import randint
```

## -Implementierung

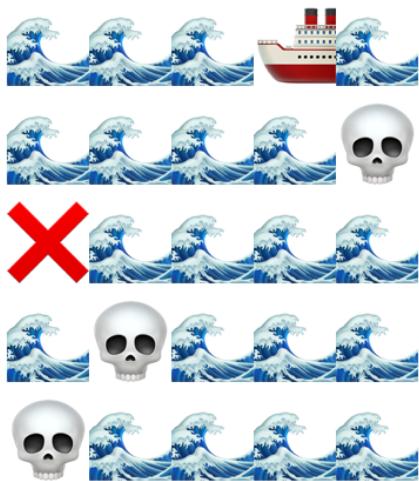
Beschreibung der Software für Benutzer:


Senden

**Eingabe der X/Y koordinaten**

**Um den Zug zu beenden muss man senden drücken**

# Its the players Turn



**Darstellung des Spielfelds**

**Totenkopf: tote Spieler Schiffe**  
**Schiff: lebendige Spieler Schiffe**  
**Kreuz: tote Gegner Schiffe**

Beschreibung für den Programmierer:

Nach der Eingabe des Spielers wurden die Koordinaten als String anerkannt was dazu führte das man nie den Gegner traff. Das wurde gelöst indem anstelle eines Integers mit einem String geprüft wird.

```
def myAttack(x,y):  
    print("the player fired at: ("+str(y)+"|"+ str(x)+")")  
    for row in db(db.enemyShipData).select():  
        if(x==str(row.x) and y==str(row.y))and(row.isAlive==True):  
            #ships[i].isAlive=False  
            print("the player landed hit on a Ship on: ("+str(y)+"|"+ str(x)+")")  
            db(db.enemyShipData.id==row.id).update(isAlive=False)
```

## -Herausforderungen und Technische Lösungen

myShip liest die Datenbank playerShipData ein und prüft ob diese voll/leer ist.

Falls die Datenbank leer ist wird sie neu gefüllt, jetzt durchläuft der code nochmals myShip und sieht nun das die Datenbank voll ist. Hier wird der inhalt der datenbank in die class Ship gefüllt die sich in der list myShips befindet.

Zur Sicherheit wird noch geprüft ob die Datenbank mehr Inhalt hat als erlaubt hat. In diesem Fall wird die ganze Datenbank gelöscht und myShip wird neu durchlaufen.

Das ganze wird für enemyShips wiederholt.

```
def myShip():
    for row in db(db.playerShipData).select():
        global counter
        counter=counter+1

    if(counter==0):
        for i in range(numberOfShips):
            myShips[i]=Ship(db.playerShipData.insert(x=randint(0,fieldSize-1), y=randint(0,fieldSize-1),isAlive=True))
            alivePlayer=numberOfShips
            myShip()

    if(counter!=0 and counter>numberOfShips):
        db.playerShipData.truncate()
        counter=0
        myShip()

    if(counter!=0):
        i=0
        for row in db(db.playerShipData).select():
            myShips[i]=Ship(row.x, row.y,row.isAlive)
            counter=0
            i=i+1
        i=0
```

feldPrint ist zuständig für das erstellen des spielfeldes, das macht es in dem es abhängig verschiedener faktoren entweder Schiff, Cross, Skull oder Wave printed, um im Browser zu funktionieren wurde alles in einen string verwandelt der mit \n jeweils die Zeilen getrennt hältt.

```
def feldPrint():
    global message
    message=""
    for i in range(fieldSize):#column
        global coloumn
        coloumn=""
        for k in range(fieldSize):#row
            for j in range(numberOfShips):
                #for<our the shipvvv
                if(myShips[j].x==i and myShips[j].y==k and myShips[j].isAlive==True):
                    coloumn=coloumn+"⛵"
                if(myShips[j].x==i and myShips[j].y==k and myShips[j].isAlive==False):
                    coloumn=coloumn+"💀"
                if(enemyShips[j].x==i and enemyShips[j].y==k and enemyShips[j].isAlive==False):
                    coloumn=coloumn+"✖"
            coloumn=coloumn[:k+1]+"
            print(coloumn)
            message=message+'\n'+ coloumn
#print(message)
```

## -Beschreibung für Kollegen

der gesammte Code wurde in python3 geschrieben, das ist wichtig zu erwähnen da vorherige, und eventuel zukünftige versionen von python anders mit der print funktion umgehen. Die zugewießenen Variablen sind **nicht** zu verändern da es sonst zu problemen führt. es ist erlaubt die Datenbanken zu entleeren, das programm erstellt automatisch neue. Der Code ist ein Auto das mit Klebeband zusammengehalten wird, löse ein Band und das ganze Auto fällt zusammen. Ein paar Worte der Weisheit:



*It just works*

Um den Code zu verstehen Schauen wir uns die Mainloop an und Arbeiten die verschiedenen Funktionen durch.

```
def index():
    inputX=request.vars['inputX']
    inputY=request.vars['inputY']
    #---as soon as one of each has lost his ships, the game will delete the list and start a new one.
    gameOverCheck()

    #beginning:declares the ship coordinates and fills player with a list to work out the turnbased system
    playerList()
    myShip()
    enemyShip()

    #prints the field
    feldPrint()

    #the game starts
    if(turnList[0].playerTurn==True):
        #players turn
        print("its the players turn")
        turnAnnouncement="Its the enemies Turn"
        #----the player now attacks
        myAttack(inputY,inputX)

        #----changes playerTurn to false so its the enemys turn
        print("the player has fired and finished his Turn")
        db.player.truncate()
        turnList[0]=Player(db.player.insert(playerTurn=False))

        return dict(message=message,turnAnnouncement=turnAnnouncement,turnInfo=turnInfo)#myShips[0].isAlive
    else:
        #enemys turn
        print("its the enemies turn")
        turnAnnouncement="Its the players Turn"
        #----the enemys AI fires into the void, hoping to hit something
        enemyAi()

        #----changes playerTurn to True so its the players turn
        print("the enemy has fired and finished his Turn")
        db.player.truncate()
        turnList[0]=Player(db.player.insert(playerTurn=True))

        return dict(message=message,turnAnnouncement=turnAnnouncement,turnInfo=turnInfo)#myShips[0].isAlive
```

```

inputX=request.vars['inputX']
inputY=request.vars['inputY']
#---as soon as one of each has lost his ships, the game will delete the list and start a new one.
gameOverCheck()

```

Zu Beginn wird der vom Spieler gegebene Input als Variable festgelegt, da der Code beim Abschicken der Daten von neu durchgelaufen wird ist es wichtig das das ganz zu Beginn passiert.

## -GameOverCheck

```

def gameOverCheck():
    global gameOver
    #checks status of game
    alivePlayer=0
    for row in db(db.playerShipData.isAlive == False).select():
        alivePlayer=alivePlayer+1
    if(alivePlayer==numberOfShips):
        gameOver=True

    aliveEnemy=0
    for row in db(db.enemyShipData.isAlive == False).select():
        aliveEnemy=aliveEnemy+1
    if(aliveEnemy==numberOfShips):
        gameOver=True

    if(gameOver==True):
        db.playerShipData.truncate()
        db.enemyShipData.truncate()
        db.firingPath.truncate()
        print("The battle is over")

```

Abhängig davon wie viele Schiffe der Gegner und der Spieler verloren haben wird alivePlayer und aliveEnemy erhöht, zugegeben der Name ist irreführend und kann gerne geändert werden. Danach wird geprüft ob die Toten Schiffe gleich der erlaubten Anzahl an Schiffen sind, falls das eintrifft wird gameOver zu True und alle Datenbanken werden gelöscht, eine neue Runde kann nun gestartet werden.

```

#beginning: declares the ship coordinates and fills player with a list to work out the turnbased system
playerList()
myShip()
enemyShip()

```

Wir behandeln nur playerList und myShip da enemyShip nur myShip mit enemy ist.

## -PlayerList

```

def playerList():
    for row in db(db.player).select():
        global counter
        counter=counter+1

    if(counter==0): #only runs if game is started for the first time
        turnList[0]=Player(db.player.insert(playerTurn=True))
        playerList()

    if(counter!=0): #uses database to get playerTurn
        for row in db(db.player).select():
            turnList[0]=Player(row.playerTurn)
        counter=0

```

In playerList wird geprüft ob die Datenbank voll ist, ist sie leer endet counter bei 0. Üblicherweise ist das zu Beginn des Spieles. Die Liste turnList wird mit der Klasse Player gefüllt der die bool playerTurn von der Datenbank bekommt. Der Code startet jetzt wieder bei playerList und läuft jetzt bei counter ungleich 0 durch, hier wird die Liste einfach nur nochmal mit der Datenbank gefüllt. Das ist **wichtig** da das Programm sonst einen Fehler ausgibt wenn es mit einer vorhandenen Datenbank den Code neu durchläuft.

## -MyShips/EnemyShips

```
def myShip():
    for row in db(db.playerShipData).select():
        global counter
        counter=counter+1

    if(counter==0):
        for i in range(numberOfShips):
            myShips[i]=Ship(db.playerShipData.insert(x=randint(0,fieldSize-1), y=randint(0,fieldSize-1),isAlive=True))
            alivePlayer=numberOfShips
            myShip()

    if(counter!=0 and counter>numberOfShips):
        db.playerShipData.truncate()
        counter=0
        myShip()

    if(counter!=0):
        i=0
        for row in db(db.playerShipData).select():
            myShips[i]=Ship(row.x, row.y, row.isAlive)
            counter=0
            i=i+1
        i=0
```

myShip liest die Datenbank playerShipData ein und prüft ob diese voll/leer ist. Falls die Datenbank leer ist wird sie neu gefüllt, jetzt durchläuft der code nochmals myShip und sieht nun das die Datenbank voll ist. Hier wird der inhalt der datenbank in die Klasse Ship gefüllt die sich in der Liste myShips befindet. Zur Sicherheit wird noch geprüft ob die Datenbank mehr Inhalt hat als erlaubt hat. In diesem Fall wird die ganze Datenbank gelöscht und myShip wird neu durchlaufen. Das ganze wird für enemyShips wiederholt.

```
#prints the field
feldPrint()
```

## -FeldPrint

```
def feldPrint():
    global message
    message=""
    for i in range(fieldSize):#column
        global coloumn
        coloumn=""
        for k in range(fieldSize):#row
            for j in range(numberOfShips):
                #for our the shipvvv
                if(myShips[j].x==i and myShips[j].y==k and myShips[j].isAlive==True):
                    coloumn=coloumn+"🚢"
                if(myShips[j].x==i and myShips[j].y==k and myShips[j].isAlive==False):
                    coloumn=coloumn+"💀"
                if(enemyShips[j].x==i and enemyShips[j].y==k and enemyShips[j].isAlive==False):
                    coloumn=coloumn+"✖"
            coloumn=coloumn[:k+1]+"""
            print(coloumn)
            message=message+'\n'+ coloumn
    #print(message)
```

feldPrint ist zuständig für das erstellen des spielfeldes, das macht es in dem es abhängig verschiedener faktoren entweder Schiff, Cross, Skull oder Wave printed, um im Browser zu funktionieren wurde alles in einen string verwandelt der mit \n jeweils die Zeilen getrennt hält.

```

#the game starts
if(turnList[0].playerTurn==True):
    #players turn
    print("its the players turn")
    turnAnnouncement="Its the enemies Turn"
    #----the player now attacks
    myAttack(inputY,inputX)

    #----changes playerTurn to false so its the enemys turn
    print("the player has fired and finished his Turn")
    db.player.truncate()
    turnList[0]=Player(db.player.insert(playerTurn=False))

    return dict(message=message,turnAnnouncement=turnAnnouncement,turnInfo=turnInfo)#myShips[0].isAlive

```

## -MyAttack/EnemyAttack(inputY,inputX)

```

def myAttack(x,y):
    print("the player fired at: ("+str(y)+"|"+ str(x)+")")
    for row in db(db.enemyShipData).select():
        if(x==str(row.x) and y==str(row.y))and(row.isAlive==True):
            #ships[i].isAlive=False
            print("the player landed hit on a Ship on: ("+str(y)+"|"+ str(x)+")")
            db(db.enemyShipData.id==row.id).update(isAlive=False)

```

Mit den Eingaben des Spielers wird nun die gesammte Datenbank der gegnerischen Schiffe durchgelaufen und auf Übereinstimmung geprüft. Wenn diese bedingung True ist wird die isAlive bool von diesem Schiff zu False, damit wird dieses Koordinaten Paar von **FeldPrint()** als rotes Kreuz dargestellt und ist zukünftig nicht mehr treffbar. EnemyAttack funktioniert mit dem selben Prinzip.

danach wird die Player Datenbank gelöscht und eine neue wird erstellt bei der playerTurn nun False ist. Das sorgt dafür, dass bei erneutem durchgang der gegner zum Zug kommt.

```

else:
    #enemys turn
    print("its the enemies turn")
    turnAnnouncement="Its the players Turn"
    #----the enemys AI fires into the void, hoping to hit something
    enemyAi()

    #----changes playerTurn to True so its the players turn
    print("the enemy has fired and finished his Turn")
    db.player.truncate()
    turnList[0]=Player(db.player.insert(playerTurn=True))

    return dict(message=message,turnAnnouncement=turnAnnouncement,turnInfo=turnInfo)#myShips[0].isAlive

```

## -EnemyAi

```

def enemyAi():
    #-----keeps its own position in mind while shooting for enemy
    for row in db(db.firingPath).select():
        global counter
        counter=counter+1

    if(counter==0): #only runs if game is started for the first time
        i=0
        for row in db(db.enemyShipData).select():
            fieldOffFire[i]=FiringPath(db.firingPath.insert(x=row.x, y=row.y))
            i=i+1
        i=0
        enemyAi()

    if(counter!=0): #uses database to ge
        i=0
        for row in db(db.firingPath).select():
            fieldOffFire[i]=FiringPath(row.x, row.y)
            i=i+1
        counter=0
    ranCord()

```

Wie bei allen Datenbanken wird hier geprüft ob schon eine vorhanden ist und ob es nötig ist eine Datenbank anzulegen. Da nur eine dumme AI auf sich selbst schießen würde, hab ich die Datenbank FiringPath erstellt. Hier werden die Koordinaten des Gegners eingefügt, um sicher zu gehen das keine Schüsse verschwendet werden werden nun in **rancord()** geschaut ob die durch Zufall entstandene Koordinaten mit einer der eigenen übereinstimmt.

Falls dem so ist wird **rancord()** neu durchlaufen.

```
def ranCord():
    x=randint(0,fieldSize-1)
    y=randint(0,fieldSize-1)

    for row in db(db.firingPath).select():
        if(x==row.x and y==row.y):
            ranCord()

    enemyAttack(x,y)
```

Die fertigen Koordinaten werden nun an **enemyAttack()** weitergeleitet. Nach Abschluss des Zuges wird auch hier wieder die Datenbank Player gelöscht und neu angelegt, jetzt wieder mit playerTurn=True.

## **-Fazit**

Ich habe gelernt einfache anwendungen in Web2py zu erstellen, und mir das Arbeiten an einem Projekt einzuteilen.

Das einzige was ich eventuell verändern wollen würde ist Code duplizierung, andere Programmierer haben mich dafür schon verhauen, da ich mir für dieses Projekt alles selbst beibringen musste hab ichs better safe than sorry gemacht.

## **-Quellen**

- <http://web2py.com/books/default/chapter/29/06/the-database-abstraction-layer>
- <https://www.youtube.com/watch?v=yKQNmutC0WY&t=162s>
- <http://www.web2py.com/init/default/download>
- <https://docs.python.org/3/library/random.html>
- [https://pre00.deviantart.net/a77d/th/pre/f/2019/054/a/6/7a056468\\_c66b\\_4f69\\_a857\\_0ebc8b1e4a38\\_by\\_meatmantires-dd0jx6h.png](https://pre00.deviantart.net/a77d/th/pre/f/2019/054/a/6/7a056468_c66b_4f69_a857_0ebc8b1e4a38_by_meatmantires-dd0jx6h.png)