GaTher3DEvo
Instruction
How to modify the software

In the following instructions we will answer the following question:

* How to add more game problems?

!!! If you would like to add a new reproduction function, mortality, neighbourhood, or resource function follow the implementation would be identical. In case of any problems create a github issue.

Step 1 Information needed for creating a new fitness problem:

* Problem matrix
* Parameteres of the fitness problem
* Number of phenotypes for a given problem.
* Is the problem dynamic? Or the problem shall not change during the simulation.

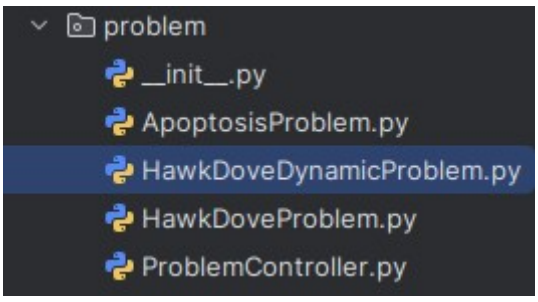
To explain the problem lets implement together the Dynamic Hawk Dove problem.
The fitness matrix looks as follows:

	$enemy\ Hawk$	$enemy\ Dove$
$player\ Hawk$	$(V-C)/2$	V
$player\ Dove$	$r*V*0.25$	$V*0.5*(r+1)$

The problem contains 2 phenotypes (Hawk and Dove), 2 parameters (V and C) and is dependent on the chosen resource function.

Step 2 Create a problem class:

The problem class shall be located in a new file named as the class in a problem directory and shall inherit from the ProblemController class.



At the top of the file the requried packages shall be imported. They may differ a bit depending on an implementation.

```
from typing import List, Dict, Optional, Tuple

from neighbourhood.NeighbourController import NeighbourController
from problem.ProblemController import ProblemController
from resource.ResourceFunctionController import ResourceFunctionController
from utils.ParamHandler import ParamHandler
```

The declare a class name inhetiting from a ProblemController class. Below the class declaration parameters and phenotype_names shall be declared.

```
class HawkDoveDynamicProblem(ProblemController):
    V_param: int
    C_param: int
    phenotype_names: Dict[int, str]
```

The new class must have 2 function.

1) __init__

```
def __init__(self, param_handler: ParamHandler, neighbour_controller: NeighbourController,
resource_function_controller: Optional[ResourceFunctionController]):
    super().__init__(param_handler, neighbour_controller, resource_function_controller)

    if self.resource_function is None:
        raise Exception('Resource function cannot be None in the following problem')

    self.supported_num_dims = [2, 3]

    self.num_phenotypes = 2
    self.phenotype_names = {0: 'Hawk',
                            1: "Dove"}
```

It is important to fill the supported_num_dim list with supported dimensions, the num_phenotypes with required number of phenotypes and the dictionary phenotype_names with the names of phenotypes.

The user uses only the names in the yaml file ale they are automatically ‘translated’ into selected numbers.

If the problem is not dynamic – remove checking if the resource function is None.

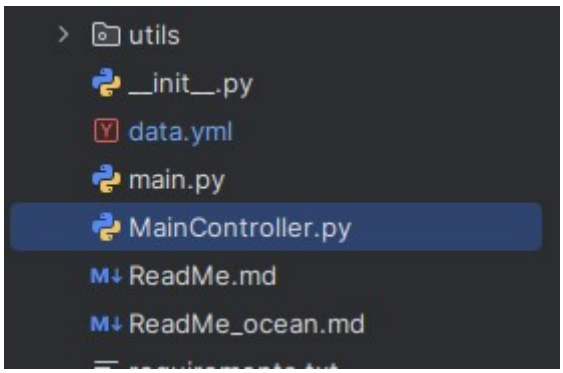
2) Fitness problem function

```
def fitness_problem(self, player: int, enemy: int, idx: Optional[Tuple[int, int] | Tuple[int, int, int]] =
None) -> float:
    r = self.resource_function.get_function_value(idx)

    if player == 0 and enemy == 0:
        return (self.V_param - self.C_param) / 2

    elif player == 0 and enemy == 1:
        return r * self.V_param * 0.25

    elif player == 1 and enemy == 0:
        return self.V_param

    elif player == 1 and enemy == 1:
        return (r+1) * self.V_param * 0.5

    else:
        raise Exception('Invalid player or enemy number')
```

This is just a fitness matrix transposed into if statements.

!!!
If the problem is not dynamic remove the r=self.resource_function line.
!!!

Step 3
Import the class name into the MainController.py file.




```
from problem.HawkDoveDynamicProblem import HawkDoveDynamicProblem
```

Step 4 file the Yaml file with the following properties.

1) Set the problem_name to class name. The names must match!!!.

```
problem_name: HawkDoveDynamicProblem
```

2) Set the problem_params to ones declared in a problem class. Assign the required values to them.

```
problem_params:
  V_param: 9
  C_param: 6
```

3) **Optional** Set the initial probabilities for the initialized game matrix. If this is not set the phenotypes will be drawn evenly.

```
initial_probability:
  Hawk: 0.5
  Dove: 0.5
```

4) **Only for the dynamic problems**

If the selected dynamic function uses the information about the alocated phenotypes (Global, Local) fill the resource_phenotypes with a chosen phenotype name.

```
resource_function_params:
  mode: time

  resource_phenotypes:
  ph1: Hawk
```

Step 5 – contribute to the github reposotory.
:)