Fishery Simulation

Documentation

Contents

Introduction

A simple simulation of a fishery, or more precisely, of a fish population which is periodically fished. The simulation can be considered to consist of four levels:

- Soil with a soil energy level.
- Vegetation which grows by consuming soil energy.
- Fish population which grows by consuming vegetation.
- Fishing yield by catching fish population.

The simulation is inspired by the book "Mathematical Bioeconomics" by Colin W. Clark. The intention is to reproduce some of the analytical curves depicting fishing yields presented in the book. To produce these curves, the simulation reports the total fish population, the amount of fish caught and the amount of underlying vegetation. The conditions of the simulation can be altered, including fish and vegetation growth rate, fishing rate and simulation length.

Simulation Structure

The simulation progresses in steps, during which each part of the simulation is updated according to the simulation logic. The simulation area consists of NxM tiles representing the ocean floor. Each tile has its own soil energy and vegetation level. The fish population consists of individual fishes which have their own food level, population level (size) and position. Only one fish can be present in one tile.

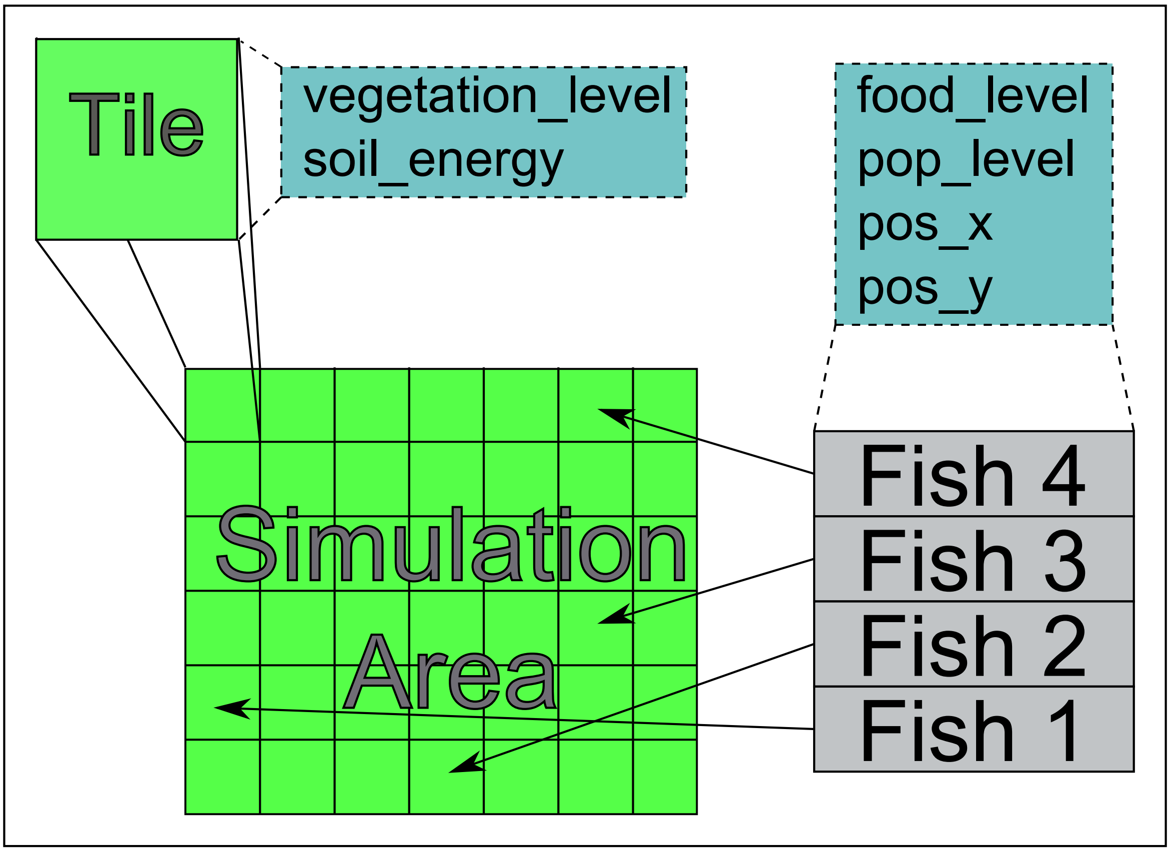

Figure 1. Structure of simulation.

Simulation Logic

Each level mentioned in the introduction has its own logic which can be modified by the program settings (see Simulation Settings for a complete list of settings).

- Soil
  - Each tile has its own soil energy.
  - Soil energy grows each turn by a constant amount defined by `soil_energy_increase_turn`.
  - Soil energy has a maximum which is defined `soil_energy_max`.
- Vegetation
  - Each tile has its own vegetation level.
  - Each turn the vegetation will consume soil energy. The amount consumed is determined by the level (size) of the vegetation according to `vegetation_consumption`.
  - If there is not enough soil energy present in the tile, the vegetation level shrinks by one.
  - If there is enough soil energy present for growth, that is `vegetation_level_growth_req` + `vegetation_consumption`[vegetation_level], the vegetation level increases by one after consuming said amount.
  - After reaching a certain vegetation level, `vegetation_level_spread_at`, vegetation will spread to adjacent tiles with no vegetation present.
- Fishes
  - Each fish has its own population (size) level and food level. There is a maximum population level defined by `fish_level_max`.
  - Each fish can occupy one tile, and each tile can have exactly one fish.
  - Every turn each fish performs a number of actions defined by `fish_moves_turn`. Possible actions are:
    - Consume vegetation at current tile.
    - No vegetation at current tile, move randomly to unoccupied adjacent tile (tiles are prioritized according to vegetation level). Consume vegetation at new tile.
    - No vegetation at current tile, move randomly to unoccupied adjacent tile (tiles are prioritized according to vegetation level). No vegetation to consume at new tile.
    - No vegetation at current tile and no move possible, do nothing.
  - The maximum amount of vegetation a fish can consume per turn is twice the amount of food it needs to survive, according to `fish_consumption`, plus `fish_growth_req`. The vegetation consumed is added to the food level of the fish.
  - Once all actions have been performed, the food level of the fish determines if it grows, stays the same, or shrinks.
    - If the food level is larger than the food needed to survive, according to `fish_consumption`, plus `fish_growth_req`, the fish grows one level and consumes said amount of food.
    - If the food level isn't large enough for growth, but large enough for survival, according to `fish_consumption`, the fish simply consumes said amount of food.
    - If the food level smaller than needed for survival, the fish loses one level and loses all food.
  - Once at max level, a fish can split (reproduce) into two. The new fish will be of level 1, while the main fish loses one level. This behavior can be turned on and off by `split_fishes_at_max`.
  - Each turn there is a random chance for a new fish to arrive, defined by `random_fishes_interval`.
- Fishing event
  - Every turn there is chance for each fish to caught, i.e. to lose one population. This chance is defined by `fishing_chance`.

Compiling the program

C

At the moment there is no make file, but compilation is simple. i.e.:

```
gcc –c ./src/help_functions.c ./src/fishery_functions.c –lm –I ./include
```

Python

Compilation of the extension for Python has been confirmed to work on Windows 7 using MSVC 2012 and Ubuntu 14.04 using gcc 4.8.4. Compilation requires:

- Python 3 with python.h
  - On Ubuntu this requires installing python3-dev, i.e. sudo apt-get install python3-dev
- C-compiler, e.g. gcc or MSVC
  - On Windows, the compilation will use whatever compiler was used to compile Python with. If you have another compiler, you have to update the appropriate settings. The compilation does work with the compiler provided with Visual Studio 2013.

Compilation is simple using the project's setup.py file:

```
python setup.py install
```

It might be needed, however, to update the paths in setup.py.

Running a Simulation

Pure C version

The C version is used by calling functions documented in Table 3. Functions in fishery_functions.c. Example code:

```
#include "fishery_functions.h"
int main(void)
{
	Fishery_Settings settings;
	Fishery *fishery;
	Fishery_Results results;
	/* Create settings. The order of the settings can be found below in Table 1. Fishery settings and their description. or in the variable SETTING_ORDER defined in fishery_functions.c. */
	int vegetation_requirements[] = {0, 1, 1, 2, 2, 3 };
	int fish_requirements[] = { 0, 1, 2, 3, 4, 5};
	settings = CreateSettings(10, 10, 80, 5, 3, 3, 10, 3, vegetation_requirements, 10,
	                          5, 1, 6, fish_requirements, 50, 1, 0.1);
	/* Create fishery. */
	fishery = CreateFishery(settings);
	/* Run simulation for 500 steps and print some results.*/
	results = UpdateFishery(fishery, settings, 500);
	printf("[%d, %d]\n", results.fish_n, results.yield);
}
```

From Python

Instead of passing the settings are simply numerical values, the Python version uses a dictionary. Example code:

```python
import fishery
#Create settings dictionary.
settings = {}
settings["size_x"] = 10
settings["size_y"] = 10
settings["initial_vegetation_size"] = 80
settings["vegetation_level_max"] = 5
settings["vegetation_level_spread_at"] = 3
settings["vegetation_level_growth_req"] = 3
settings["soil_energy_max"] = 10
settings["soil_energy_increase_turn"] = 3
settings["vegetation_consumption"] = [0, 1, 1, 2, 2, 3]
settings["initial_fish_size"] = 10
settings["fish_level_max"] = 2
settings["fish_growth_req"] = 2
settings["fish_moves_turn"] = 5
settings["fish_consumption"] = [0, 1, 2, 3, 4, 5]
settings["random_fishes_interval"] = 30
settings["split_fishes_at_max"] = 1
settings["fishing_chance"] = 0.1
#Create fishery.
fishery_id = fishery.MPyCreateFishery(settings)
#Run simulation for 500 steps and print results.
results = fishery.MPyUpdateFishery(fishery_id, 500)
print(results)
#Free memory.
fishery.MPyDestroyFishery(fishery_id)
```

Fishery Results

Each run of a simulation returns the following numerical values:

```
Total fish population size, all fish population encountered during simulation,
Total yield, i.e. fish population caught during fishing events,
Total vegetation size, i.e. all vegetation encountered during simulation,
Standard deviation of fish population per turn, calculated from end of each step,
Standard deviation of yield per turn, calculated from end of each step,
Standard deviation of vegetation per turn, calculated from end of each step,
Length of simulation in steps, TO BE REMOVED?
Debugging variable, TO BE REMOVED
Fishing chance during run, TO BE REMOVED
```

Simulation Settings

Table 1. Fishery settings and their description.

Setting name	Setting description	Type	Limitations
size_x	Width of simulation area.	integer	Memory and simulation execution time. Program only accepts 1-1000.
size_y	Height of simulation area.	integer	Memory and simulation execution time. Program only accepts 1-1000.
initial_vegetation_size	Number of tiles with vegetation at beginning.	integer	Larger than 0 but smaller than total simulation area.
vegetation_level_max	Maximum level of vegetation growth.	integer	Program only accepts values between 1 and 100.
vegetation_level_spread_at	Vegetation level at which vegetation spreads to adjacent tiles.	integer	Larger than 0 but not larger than vegetation_level_max.
vegetation_level_growth_req	Soil energy required for vegetation growth.	integer	Program only accepts values between 1 and 100.
soil_energy_max	Maximum amount of soil energy per tile.	integer	Program only accepts values between 1 and 100.
soil_energy_increase_turn	Amount by which soil energy is increased each turn.	integer	Program only accepts values between 1 and 1000.
vegetation_consumption	Array of soil energy consumed at each vegetation level.	array of integers	Array size equal to vegetation_level_max. Contains meaningless entry for vegetation level 0.
initial_fish_size	Number of fishes present at beginning.	integer	Must be smaller than simulation area.
fish_level_max	Maximum level of fish.	integer	Program only accepts values between 1 and 100.
fish_growth_req	Food level required for fish growth.	integer	Program only accepts values between 1 and 100.
fish_moves_turn	Number of actions for each fish each turn.	integer	Program only accepts values between 1 and 100.
fish_consumption	Array of vegetation consumed at each fish level.	array of integers	Array size equal to fish_level_max. Contains meaningless entry for fish level 0.
random_fishes_interval	Chance of a fish randomly arriving in the simulation each turn.	integer	Between 0 and 100.
split_fishes_at_max	Boolean for fish splitting at max fish level.	integer	0 for no splitting, 1 for splitting at max fish level.
fishing_chance	The chance for each fish to be caught each turn.	double	The program accepts values 0.0-1.0.

Source code documentation

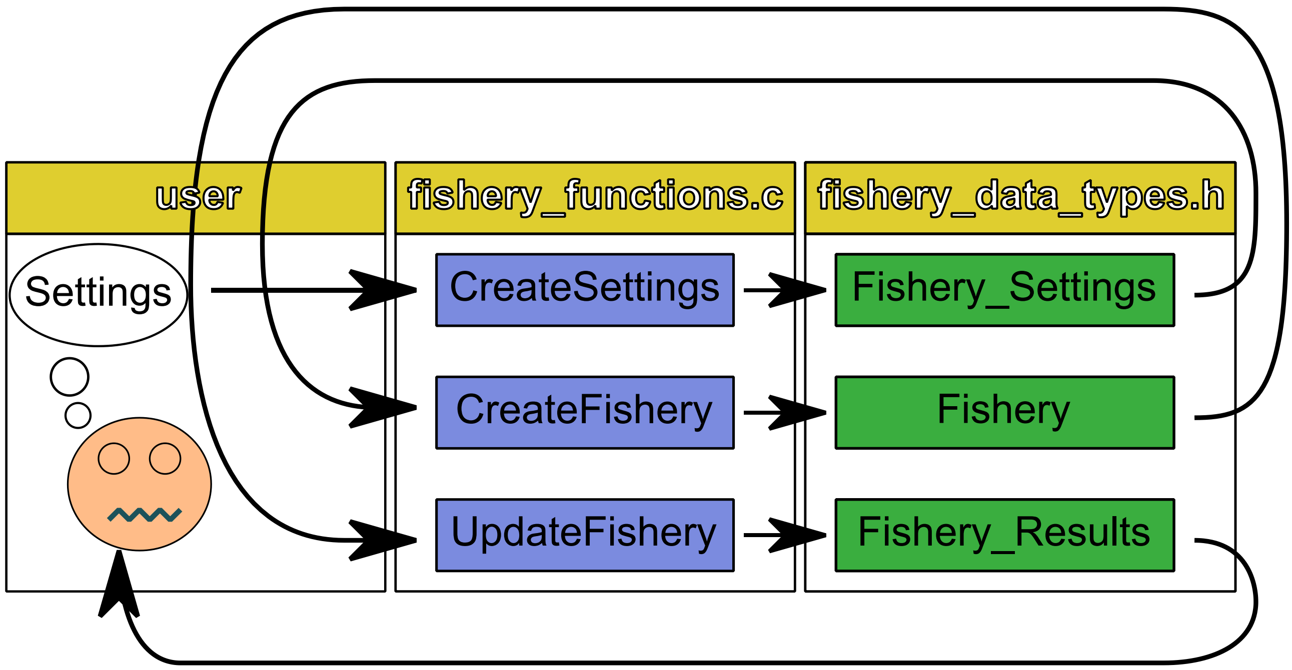

Figure 2. Representation of the internal structure of the simulation.

fishery_data_types.h defines several data structures which are used to house the simulation and its constituent parts. These structures are manipulated according to the simulation logic, which is implemented in fishery_functions.c. Finally, some supporting functions have been put in separate file, help_functions.c, for convenience.

fishery_data_types.h

Definitions for the typedef structs used by the simulation.

- `typedef struct llist_node {} LList_Node`
  - `struct llist_node *next;`
  - `void *node_value;`
- `typedef struct fish_pool {} Fish_Pool`
  - `int pop_level`
  - `int food_level`
  - `int pos_x`
  - `int pos_y`
- `typedef struct tile {} Tile`
  - `int vegetation_level`
  - `int soil_energy`
  - `Fish_Pool *local_fish`
- `typedef struct fishery_settings {} Fishery_Settings`
  - `int size_x`
  - `int size_y`
  - `int initial_vegetation_size`
  - `int vegetation_level_max`
  - `int vegetation_level_spread_at`
  - `int vegetation_level_growth_req`
  - `int soil_energy_max`
  - `int soil_energy_increase_turn`
  - `int *vegetation_consumption`
  - `int initial_fish_size`
  - `int fish_level_max`
  - `int fish_growth_req`

        - int fish_moves_turn
        - int *fish_consumption
        - int random_fishes_interval
        - int split_fishes_at_max
        - double fishing_chance
    - typedef struct fishery {} Fishery
        - Tile *vegetation_layer
        - LList_Node *fish_list
        - int fishery_id
        - Fishery_Settings *settings
    - typedef struct fishery_results {} Fishery_Results
        - int yield
        - int fish_n
        - int vegetation_n
        - int debug_stuff
        - double yield_std_dev
        - double fish_n_std_dev
        - double vegetation_n_std_dev
        - int steps

help_functions.c

Contains various help functions needed in the implementation of the fishery simulation.

Table 2. Functions in help_functions.c.

LList_Node *LListCreate(void) Creates empty linked list. Returns pointer to first node.	
int LListIsEmpty(LList_Node *root) Checks if linked list is empty, e.g. the first node is empty and does not link to another node. Returns 1 if empty, 0 otherwise.	
LList_Node *LListAdd(LList_Node *root, void *node_value) Adds node pointing to provided value.	
root	Start of linked list.
node_value	Pointer to element to be stored in list.
void *LListPop(LList_Node *root, const void *node_value, int (*CompareValues)(const void *value1, const void *value2)) Removes first node containing node_value encountered in list. Node values are compared using CompareValues. If node_value is NULL, the last element of the list is removed. Returns a pointer to the value of the removed node, or NULL if no matching node is found.	
root	Start of linked list.
node_value	Value of node to be removed from list.
CompareValue	Function which compares node values. Returns 1 if a match, 0 otherwise.
found_node_value	Return value. Pointer to value of removed node.
void LListDestroy(LList_Node *root, void (*FreeValue)(void *node_value)) Destroys and frees memory of a linked list.	
root	Pointer to start of linked list.
FreeValue	Function which frees memory of node value.
int GetNewCoords(int cur_coords, int radius, int size_x, int size_y, Fishery *fishery) Generates new, random coordinates for a fish pool. New coordinates are checked to be not out of bounds and to not contain any fish pools. Coordinates are prioritized according to vegetation level. Returns -1 if no coordinates can be generated.	
cur_coords	Current coordinates of fish pool in one dimension.
radius	Largest allowed distance from current coordinates to new coordinates.
size_x	Width of vegetation layer in fishery simulation.
size_y	Height of vegetation layer in fishery simulation.
fishery	Pointer to initialized fishery simulation.
new_coords	New coordinates in one dimension. -1 if no possible coordinates are available.

fishery_functions.c

Contains all functions for manipulating fishery simulations.

Table 3. Functions in fishery_functions.c.

`int ValidateSettings(Fishery_Settings settings, int output_print)`	
Validates fishery settings, i.e. checks for impossible values which would break the simulation. Returns 1 on success, 0 if invalid settings.	
settings	Settings to be validated.
output_print	If 1, prints output regarding invalid settings. If 0, no output is printed.
`Fishery_Settings CreateSettings(int size_x, int size_y, int initial_vegetation_size, int vegetation_level_max, int vegetation_level_spread_at, int vegetation_level_growth_req, int soil_energy_max, int soil_energy_increase_turn, int *vegetation_consumption, int initial_fish_size, int fish_level_max, int fish_growth_req, int fish_moves_turn, int *fish_consumption, int random_fishes_interval, int split_fishes_at_max, double fishing_chance)`	
Creates Settings structure which is used in functions which manipulate the fishery. Returns settings structure.	
`Fishery *CreateFishery(Fishery_Settings settings)`	
Initializes and returns fishery according to given fishery settings.	
settings	Settings for fishery, created with CreateSettings function.
`Fishery_Results UpdateFishery(Fishery *fishery, Fishery_Settings settings, int n)`	
Progresses the fishery n steps using the given settings. Returns the results of the simulation steps in a Fishery_Result structure. The results contain total vegetation level, total fish population and total fishing yield, as well as their standard deviations. See the Fishery_Results structure for details.	
fishery	Initialized or progressed fishery.
settings	Settings for fishery.
n	Number of steps to progress the simulation.
results	Results of simulation run, including total amount of fish present as well as total fishing yield.
`void UpdateFisheryVegetation(Fishery *fishery, Fishery_Settings settings)`	
Increases soil energy and grows the vegetation layer as necessary.	
fishery	Initialized or progressed fishery.
settings	Settings for fishery.
`void UpdateFisheryFishPopulation(Fishery *fishery, Fishery_Settings settings)`	
Updates the fish population of the fishery simulation. This includes growing fish pools, moving fish pools around in search of food and consuming vegetation. Also generates new fish pools.	
fishery	Initialized or progressed fishery.
settings	Settings for fishery.
`int FishingEvent(Fishery *fishery, Fishery_Settings settings)`	
Releases the fishing boats! Based on the fishing_chance, each fish has a random chance of being fished. Returns the yield of the fishing event, i.e. total amount of fish population level lost.	
fishery	Initialized or progressed fishery.
settings	Settings for fishery.
yield	Fish population lost (caught) during event.
`void DestroyFishery(Fishery *fishery)`	
Frees memory used by fishery simulation.	
fishery	Initialized or progressed fishery.
settings	Settings for fishery.

Python interface / functions

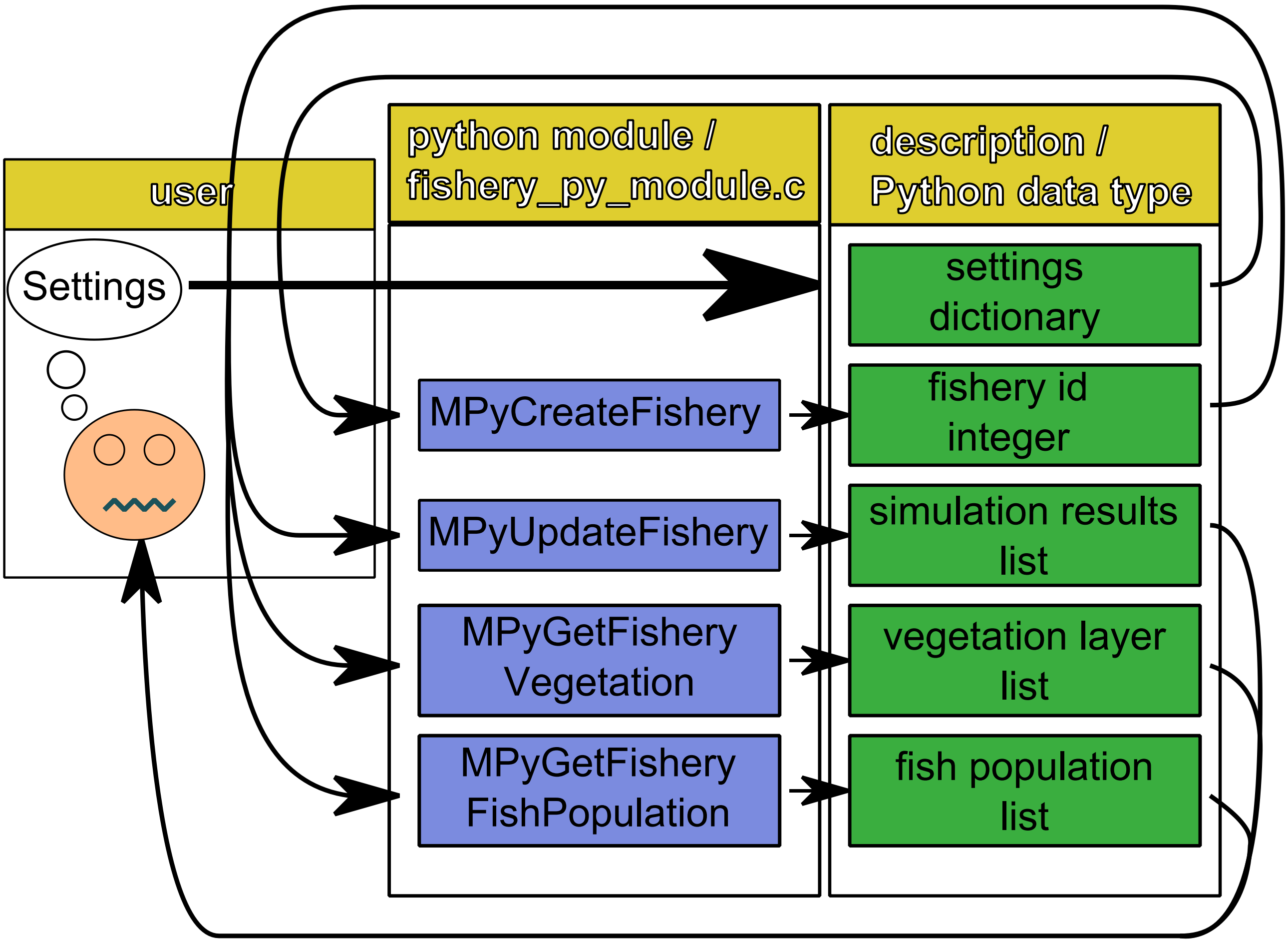

Figure 3. Structure of the Python interface.

Table 4. Functions available in the Python interface.

MPyCreateFishery(settings)

```
Initializes fishery simulation using the provided settings and assigns it a unique ID.

Keyword arguments:
settings            Dictionary of settings. Setting names are defined in
                    fishery_functions.c and in Table 1. Fishery settings and their description.

Returns:
fishery_id          A unique ID for the simulation.
```

MPyUpdateFishery(fishery_id, n) Progresses the fishery simulation n steps, returning the results. The results of the simulation are returned as a list containing the following values: ○ total number of fish ○ fishing yield ○ total vegetation level ○ standard deviation of fish per step ○ standard deviation of fishing yield per step ○ standard deviation of vegetation per step ○ steps progressed ○ debugging variable ○ fishing chance of simulation Keyword arguments: fishery_id ID of simulation. n Number of steps to progress simulation. Returns: results Results of the simulation run. See above for details.
MPyDestroyFishery(fishery_id) Removes simulation and frees memory used by it. Keyword arguments: fishery_id ID of simulation. Set -1 to destroy all simulations.
MPyGetFisheryVegetation(fishery_id) Returns the vegetation layer of a simulation. Keyword arguments: fishery_id ID of simulation. Returns: vegetation_layer Python list of the vegetation levels.
MPyGetFisheryFishPopulation(fishery_id) Returns the fish population of a simulation. Keyword arguments: fishery_id ID of simulation. Returns: fish_population Python list of fish population levels and positions.