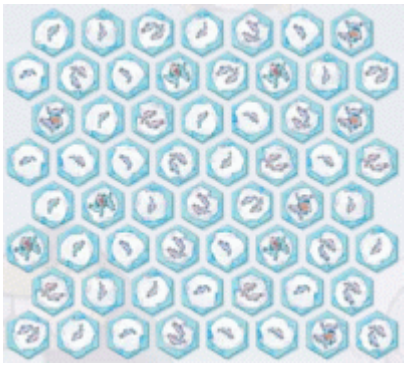
Project FISHES

Game instruction

The board consists of hexagonal fields, which represent ice floes. Size of the board can be different for each game. Each player has a set of penguins which can jump from on ice floe to another. On each field a penguin can find 1, 2 or 3 fishes. The aim is to capture maximum number of fishes.



There are two phases of the game:

1. Placing the penguins
    a. Players are informed how many penguins each player has
    b. The first player places one of his penguin on any unoccupied ice floe containing exactly one fish.
    c. Then, in turn other players place their penguins (one penguin in a turn and only on floe with exactly one fish) until all penguins are placed.
2. Playing the game
    a. Game is played in turns.
    b. In his turn, a player chooses one of his penguins. Then, he collects the fishes that the chosen penguin is standing on and makes movement of this penguin. After collecting the fishes, the field representing ice floe is removed from the board which makes a gap on the board.

   Penguin can be moved in any direction as far as the player wants but only over unoccupied ice floes. Direction of the movement cannot be changed during the movement, only moves in a straight line are allowed. Crossing ice floes with other penguins or gaps is not allowed - penguin must immediately stop when he meets other penguin or a gap on his way. The game ends when no player can move any penguin or after given number of turns.

Your task

Your aim is to deliver a program that will play the game on your behalf.
Program should accept command line parameters:

- `phase` can takes value `placement` or `movement`;
- `penguins=N`, where N is a number of penguins that player has; This parameter can only be used if `phase=placement`;
- `inputboardfile` - name of file that contains current board
- `outputboardfile` - name of file in which the new board should be stored

Examples:
penguins.out phase=placement penguins=3 inputboard.txt outpuboard.txt

penguins.out phase=movement inputboard.txt outputboard.txt
If `phase=placement`, then the program is supposed to place **one** penguin on the board and save the board to the output file. The program should only place penguin if they are not all placed on the board, assuming that the number of penguins to be placed is defined by `penguins` parameter. Therefore, the following sequence of runs should place all penguins for two players and store the final board in file outpuboard.txt:
player1.out phase=placement penguins=2 board0.txt board1.txt
player2.out phase=placement penguins=2 board1.txt board2.txt
player1.out phase=placement penguins=2 board2.txt board3.txt
player2.out phase=placement penguins=2 board3.txt board4.txt

If `phase=movement`, then the program is supposed to play one turn, i.e., collect fishes, remove related piece of floe and move one penguin. The following sequence of runs arranges 2 turns for two players:
player1.out phase=movement board4.txt board5.txt
player2.out phase=movement board5.txt board6.txt
player1.out phase=movement board6.txt board7.txt
player2.out phase=movement board7.txt board8.txt

The program should also be ready to be compiled for an interactive mode (use preprocessor directives). In the interactive mode the program should ask a user for decisions about penguins placement and movements. The program should work in a loop, each loop should be responsible for one action of a user. In each loop (turn) the program should read input file and write to the output file.

It is group project, however, each student is expected to be able to explain any part of your code. Contribution of each student must be clear and reflected in the source files. Each group is expected to deliver report containing information about each group member contribution (in details) and short description of the main algorithm.

Assessment

- quality of the code (use of proper data types, formatting, comments),
- structure of sources (division into files),
- fulfillment of the requirements,
- final result compared to other programs*.

*At the last tutorial we will organize a tournament. Your program will fight against programs prepared by other groups.

Expected schedule (7 meetings)

1. Introduction, preliminary discussion, kick-off. Establish your team. Use flowcharts to sketch the general concept. Design a structure of input/output files. Organize your work (git recommended).
2. Decision on the file formats. Preliminary code for interactive mode: main loop, interaction with user.
3. Structure of the project - files, main functions.
4. Data structures for board, printing board on the screen.
5. Handling the files.
6. Algorithm, final data structures.
7. Final tournament, assessment.

Remember - it's better to have simple solution that will work, then the most sophisticated algorithm but not working. Try to work incrementally, be agile!

Additional issues: think how you can prevent cheating by your opponent.