Lab Assignment 6

CS 361 – Principles of Programming Languages I

Winter 2022

The goal of this lab is to program the game Connect Four. See [1] if you are unfamiliar with the game.

You are already given a file *main.cpp* (which you may not change). You have to implement the remaining of the game. Next to the *main.cpp*, your program will contain the files *game.h*, *game.cpp*, *players.h*, and *players.cpp*.

The file *game.h* defines the class *Game*, an enumeration *GameState* which represents the possible game states, and an enumeration *BoardField* which represents the possible state of a field of the game-board. Declare these enumerations as so-called *scoped* enumerations, i. e., with a `class`-keyword. The file *players.h* defines an abstract class *Player*, a class *HumanPlayer*, and a class *AiPlayer*. The classes *HumanPlayer* and *AiPlayer* are subclasses of the class *Player*. Make sure that the header files only contain declarations. You may only implement class members in the corresponding *.cpp*-files.

Player Classes

Player. The abstract class *Player* defines the interface which is used by the *Game* class. It must contain an abstract function `int getNextTurn(const Game&)` which takes as parameter the current game (to read the game state and board) and returns an integer in the range from 1 to `Game::BoardWidth` (inclusive) representing the column in which a player puts their next stone.

HumanPlayer. Represents a human player. Its function *getNextTurn* reads an integer from the terminal and returns it (no matter what it is).

AiPlayer. Represents a non-human player. Its function *getNextTurn* first determines in which column the player can add its next stone. Then, out of all possible columns, one is picked randomly and returned. It also outputs the selected column together with a newline-character to the terminal. For example, if column 3 is selected, the output is `"3\n"`. This is the only place where your code is allowed to output anything to the terminal.

Game Class

The class *Game* contains the logic for the game. It determines if turns of players are valid, keeps track of the board, determines if a player has won, and updates its state accordingly. The class has the public members listed below. You are are not allowed to add any other public members (with exception of a destructor), but you may add additional private members.

`BoardWidth`. A static constant with the value 7.

`BoardHeight`. A static constant with the value 6.

Note: The declarations of *BoardWidth* and *BoardHeight* are the only places where you are allowed to hard-code their values. Use these constants everywhere else. One should be able to just change these constants and the program should still work with the updated width/height.

`Game(Player &p1, Player &p2)`. Constructor which takes the players of the game as arguments.

`BoardState getState() cost.` A function which returns the current state of the game.

`bool isRunning() const.` A function which returns *true* if the game is still running, or *false* if the game concluded with either a draw or a player winning.

`BoardField operator() (int x, int y) const.` Overrides the ()-operator. Returns the state at the board at the given coordinate. The top left element has coordinate $(0, 0)$. Returns *Empty* if the coordinate is out of range of the board.

`void nextTurn().` Function which performs the next turn. To do so, the *Game* class calls the function *getNextTurn* of the current player. If the return value is invalid, nothing happens. In case of a valid return value, the function processes the move of the player. That is, it updates the board, determines if the current player won or if a draw was reached, and updates the game state accordingly.

Submission

For your submission, upload a single zip-file to Canvas containing the following files:

- the files *game.h* and *game.cpp*, and
- the files *players.h* and *players.cpp*.

This is an individual assignment. Therefore, a submission is required from each student.

Deadline: Sunday, March 6, 11:59 p.m.

References

1. Wikipedia, *Connect Four*, Wikipedia, The Free Encyclopedia, `https://en.wikipedia.org/Connect_Four`.

Checklist

- Implement members only in *.cpp*-files.
- *Player::getNextTurn()* is abstract with *Game* as parameter.
- *HumanPlayer* and *AiPlayer* are subclasses of *Player*.
- *getNextTurn()* returns an integer in the range from 1 to 7.
- *HumanPlayer::getNextTurn()* reads an integer from the terminal and returns it.
- *HumanPlayer::getNextTurn()* may not create any output!
- *AiPlayer::getNextTurn()* determines in which column the player can add its next stone.
- *AiPlayer::getNextTurn()* out of all possible columns, one is picked randomly and returned.
- *AiPlayer::getNextTurn()* outputs column and line break.
- Enumeration *BoardField* and *GameState* declared as scoped (with `class`-keyword).
- *Game* contains no additional public members.
- *BoardWidth* and *BoardHeight* are constants, no hard-coding elsewere.
- *Game* constructor implemented.
- *BoardState* and *IsRunning* implemented.
- *Game::operator()* returns state of board (top-left = 0,0).
- *Game::operator()* returns *Empty* if coordinates are out of range.
- *Game::nextTurn()* calls *Player::getNextTurn()*.
- *Game::nextTurn()* does nothing if the return value of *getNextTurn()* is invalid.
- *Game::nextTurn()* updates the board.
- *Game::nextTurn()* determines if the current player won or if a draw was reached.
- *Game::nextTurn()* updates the game state.