Filip Segota

CSC321

Dr. Cathy Bareiss

**Connect 4**

**Problem Summary**

* Develop the connect 4 game.

**Implementation Requirements**

* Two players able to play the game.
* Players have name, id, and color.
* Have the board that can change and access its fields, check if the move is legal, and check if the game is over.
* Driver that will run the game using the instance of the Game class.
* Game can run the game, start it, make a move, finish the game, and everything else required for these functions.
* GUI platform
* Two patterns (Iterator, interface in my case)

**Use Case Diagram**

Diagram

Description automatically generated

**Start the game**

Actors: Player

Oval: Start Game (start)

Requirements:

* Any Java device with text screen and keyboard capabilities
* Graphics and “mouse capabilities”
* (2 computers mode)
* Optional undo option
* Optional wining statistics

Start Game (start) Narrative:

Actors:

* Player 1
* Player 2

Goals:

* Initialize board to empty
* Get names
* Assign players
* (Set up the two computers mode (if playing 2 computers mode))

Assumptions/Preconditions: none (because this is the initialization of the system)

Triggers:

* Program launched

End condition:

* After all goals have been reached

Post condition:

* Players are identified and ordered
* (Able to play 2 computers mode (if playing 2 computers mode))
* Current player is set

Flow of events:

* Board is created and each spot is initialized to empty
* Player 1 is created, with name and color
* Player 2 is created, with name and color
* Current player is set to either player 1 or player 2
* (Two computers establish a communication channel (If playing 2 computers mode))

**Make a move**

Actors: Player

Oval: Make a move (move)

Requirements:

* Only current player can make a move
* Can’t make an illegal move, check if move is illegal (Can’t make a move if column is full)
* After each move, check if the game is finished (win or draw)

Make a move (move) Narrative:

Actors:

* Current player (Either player 1 or player 2)

Goals:

* (Check if the communication between computers is still on (If playing 2 computers mode))
* Get current player input
* Check if move is illegal (If it is, display a message about the problem)
* Put the thing on the board
* Check if game is finished
* If game is finished, display a message with the result (Player 1 won, it’s draw, etc.)
* If game is not finished, switch the current player

Assumptions/Preconditions:

* (The communication between computers is still on (If playing 2 computers mode))
* Game is started
* Game is not finished

Triggers:

* Start game (start) occurred
* Make a move (move) occurred and the game is not finished

End conditions:

* After all goals have been reached
* (The communication between computers is interrupted in some way (If playing 2 computers mode))

Post conditions:

* (If the communication between computers is interrupted, display the message (If playing 2 computers mode))
* If the move was illegal, the message is displayed
* If the move is not illegal, current player made a move and the board changed
* It’s checked if game is finished or not
* If the game is not finished, current player has been changed
* If the game is finished, the message with the result is displayed

Flow of events:

* (Check if the communication between computers is still on (If playing 2 computers mode))
* (If the communication between computers is interrupted in some way, display the message (If playing 2 computers mode))
* Current player put the input
* Check if move is illegal
* Display the message if the move is illegal (Still under this make a move until player makes legal move)
* Change the board according to the input (If input is legal)
* Check if game is finished
* Switch the current player (If game is not finished)

Alternative flow:

* (Check if the communication between computers is still on (If playing 2 computers mode))
* (If the communication between computers is interrupted in some way, display the message (If playing 2 computers mode))
* Current player put the input
* Check if move is illegal
* Display the message if the move is illegal (Still under this make a move until player makes legal move)
* Change the board according to the input (If input is legal)
* Check if game is finished
* Display a message (If game is finished)

|  |
| --- |
| **Player** |
| -id: int  -name: String  -color: String |
| +Player(int, String, String)  +getId(): int  +getName(): String  +getColor(): String  +setId(int): void  +setName(String): void  +setColor(String): void |

|  |
| --- |
| **<<Interface>>**  **BoardInterface** |
|  |
| +getPosition(int, int): int  +setPosition(int, int, int): void  +isOver(): boolean  +isWin(): boolean  +isDraw(): boolean  +display(): String  +checkDraw(): boolean  +checkWin(int): boolean |

**UML Diagram**

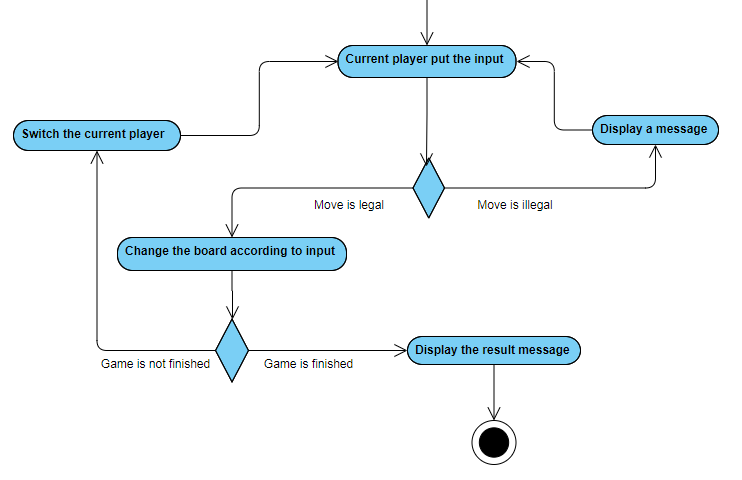
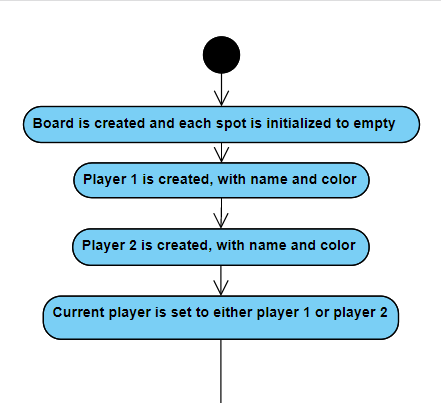
|  |
| --- |
| **Board** |
| +ROW: int  +COL: int  -field: int[][]  -end: boolean  -win: boolean  -draw: boolean  -iter: BoardIterator |
| +Board()  +getPosition(int, int): int  +setPosition(int, int, int): void  +isOver(): boolean  +isWin(): boolean  +isDraw(): boolean  +display(): String  +placeMove(int, int): void  +checkLegal(int): boolean  +checkDraw(): boolean  +checkWin(int, int, int): boolean |

|  |
| --- |
| **Game** |
| -b: Board  -p1: Player  -p2: Player  -cur: Player |
| +Game(String, String, String, String)  +Game()  +getCur(): String  +getColor(int): String  +makeMove(): void  +makeMove(int): int  +display(): String  +getBoard(): Board  +gameOver(): String  +end(): boolean  +switchPlayer(): void  +inputColumn(): int |

|  |
| --- |
| **<<Interface>>**  **Iterator** |
|  |
| +hasNext(): boolean  +next(): Object |

|  |
| --- |
| **BoardIterator** |
| -dir: int  -x: int  -y: int  -cur: int  -field: int[][] |
| +BoardIterator(int[][], int, int, int)  +changeX(int): int  +changeY(int): int  +hasNext(): boolean  +next(): Integer |

**Activity diagram**

****

(If the player connects 4, or if it’s a draw, game is finished)

**Testing Plan/Report**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case** | | | **Expected Output** | | | **Actual Output** |
| **Testing of classes** | | | | | | |
| **Player** | | | | | | |
| Player player1 = new Player(1, “Filip”, “Red”);  System.out.println(player1.getId());  System.out.println(player1.getName());  System.out.println(player1.getColor());  player1.setId(2);  player1.setName(“Mark”);  player1.setColor(“Green”);  System.out.println(player1.getId());  System.out.println(player1.getName());  System.out.println(player1.getColor()); | | | “1”  “Filip”  “Red”  “2”  “Mark”  “Green” | | | “1”  “Filip”  “Red”  “2”  “Mark”  “Green” |
| **Board** | | | | | | |
| Board b1 = new Board();  b1.setPosition(2, 2, 3);  System.out.println(b1.getPosition(2, 2));  System.out.println(b1.isOver());  System.out.println(b1.isWin());  System.out.println(b1.isDraw());  System.out.println(b1.display());  b1.placeMove(0, player1);  System.out.println(b1.getPosition(5, 0));  System.out.println(b1.checkLegal(2));  System.out.println(b1.checkDraw());  System.out.println(b1.checkWin(2));  b1.placeMove(0, player1.getId());  b1.placeMove(0, player1.getId());  b1.placeMove(0, player1.getId());  System.out.println(b1.checkWin(2));  System.out.println(b1.isWin());  System.out.println(b1.isOver()); | | “3”  false  false  false  //display the board  “2”  true  false  false  true  true  true | | | “3”  false  false  false  //display the board  “2”  true  false  false  true  true  true | |
| **Bad Cases** | | | | | | |
| Incompatible type - passing the incompatible argument to the method (ex. Int instead of String, String instead of Player,...)  Calling a method of the wrong class (ex. Calling a method of Player class with Board instance)  No suitable constructor found (ex. Not given enough argumetns to the constructor)  Trying to access out of index | player1.setId(“String”);  b1.getName();  Player p2 = new Player();  b1.setPosition(100, 100, 2); | | |  | | |

**Estimated Time**

* The coding part should take about 3-4 hours and creating an GUI should take around 15 hours.

**Identification of Outside Resources**

* None.

**Potential Security Risks**

* Can’t see any security risks (the game is too simple).
* Provided some form of security by handling exceptions.

**Potential Ethical Issues/Ethical Report**

* Can’t see any potential ethical issues. The game is too simple to create ethical problems.

**Future Improvements**

* Finding a better way to check for win in Board class (or at least get rid of ones that are unnecessary). - DONE
* Creating a log of all the players and their records
* Selecting a player from a log file and playing as him/her
* Creating a log of all the games, which would include date/time, outcome, and how long it lasted.
* Creating an option for two computers game (One player plays on one, and other plays on other).
* Making a better GUI

**Lessons Learned**

* Creating and using many diagrams, like use-case, uml, and activity diagrams.
* Learning and implementing great-code writing (Writing a driver, classes, and so on).
* Learning and using more about phases of program/game developing.
* Working with iterators
* Using constants
* Creating GUI
* Creating patterns (iterator, interface)