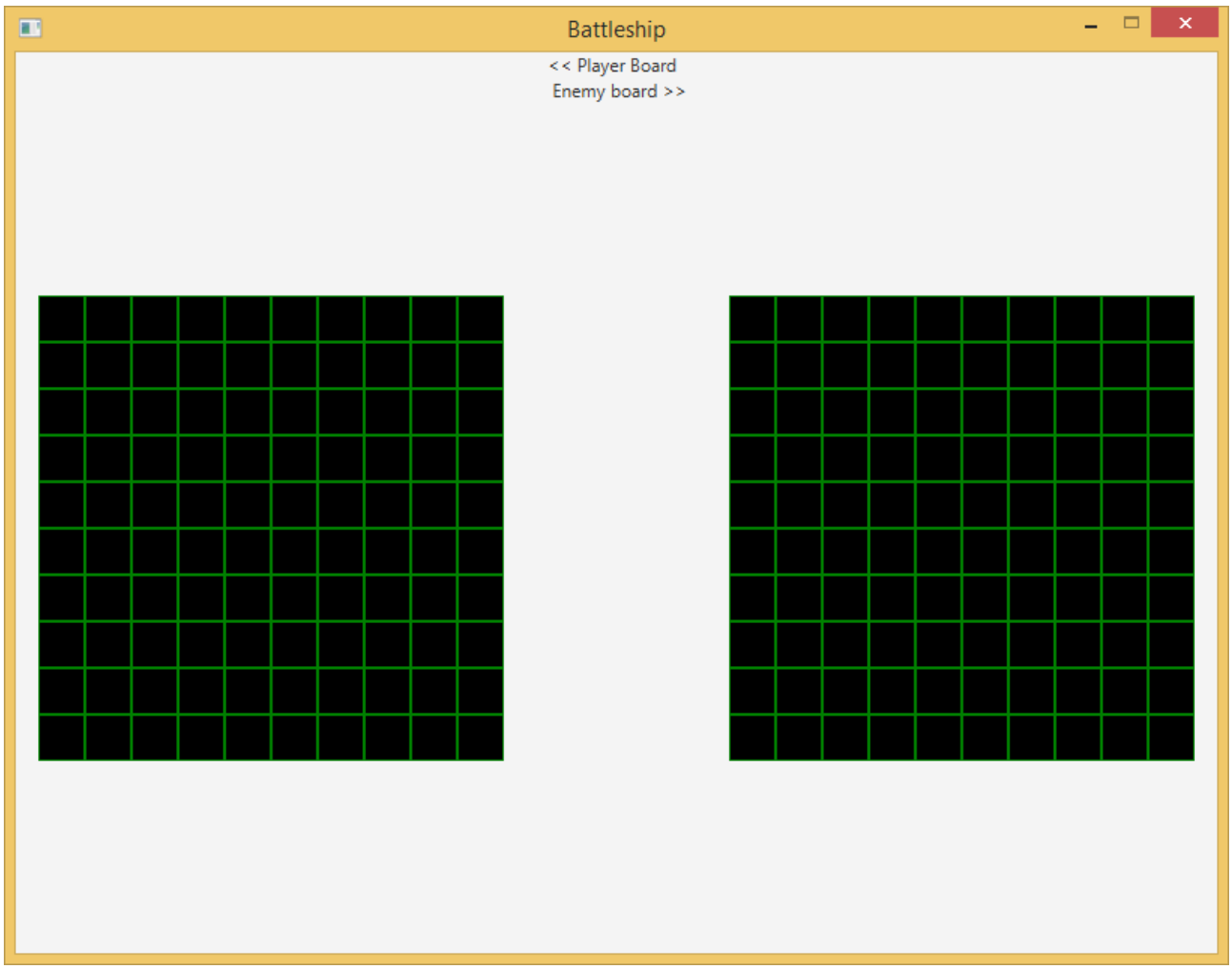
Politechnika Świętokrzyska w Kielcach
Wydział Elektrotechniki, Automatyki i Informatyki

Programowanie obiektowe (Java) - Projekt

Temat: Statki

Autor: **Piotr Czajkowski**
Paweł Czyżewski

1. Opis działania.



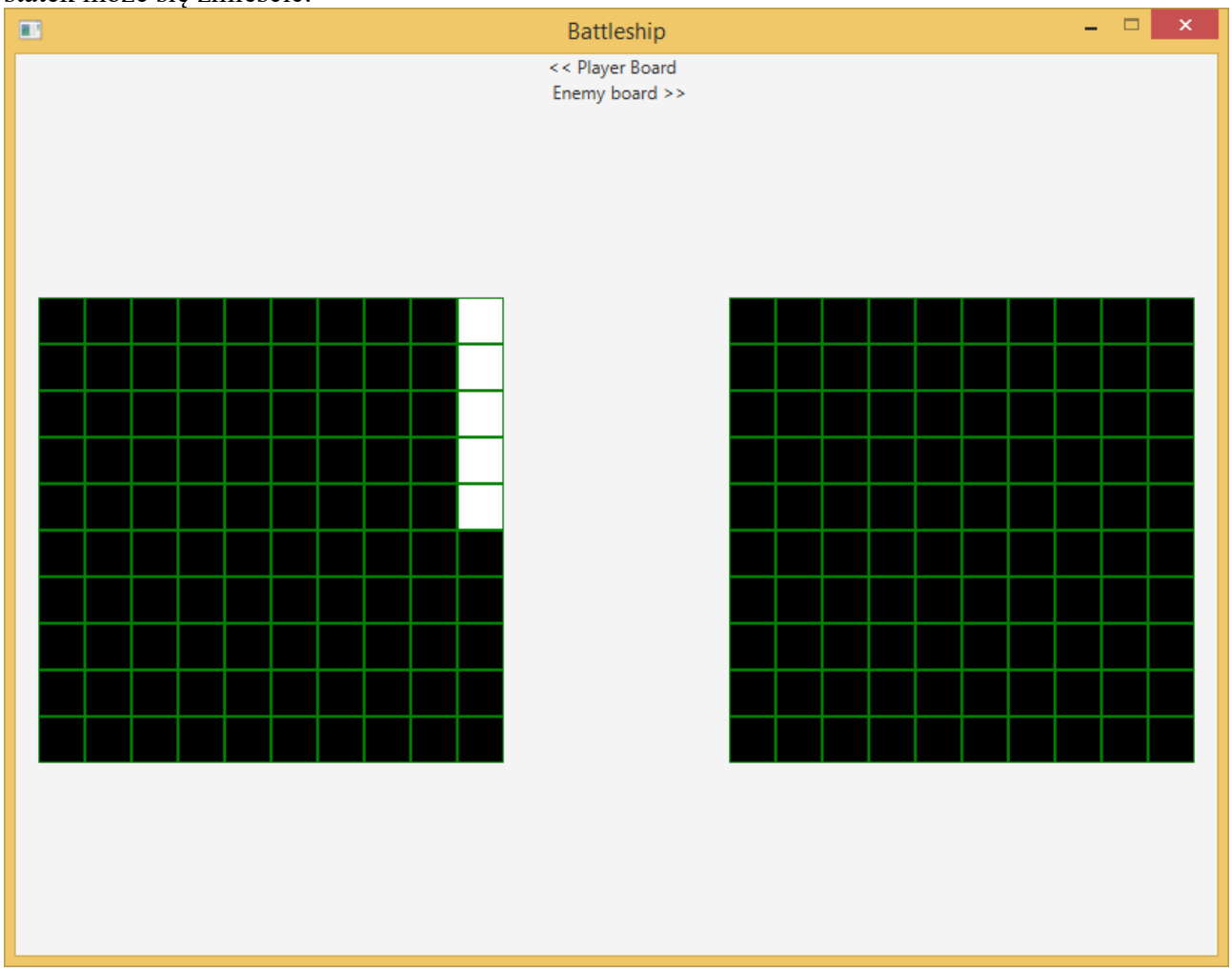
W oknie aplikacji znajdują się 2 plansze, lewa dla gracza, która służy do stawiania statków oraz prawa służąca do „strzelania” w statki przeciwnika.
Gra zaczyna się od ustawienia statków o wielkości 5, 4, 3, 2 na mapie. Po ustawieniu ich można zacząć wybierać pola na prawej planszy w celu zatopienia statków przeciwnika.

Biały kwadrat symbolizuje puste miejsce w które się strzeliło.
Chabrowy kwadrat symbolizuje miejsce w którym postawiło się własny statek.
Czerwony kwadrat symbolizuje miejsce gdzie znajduje się statek, własny lub przeciwnika, który został trafiony.

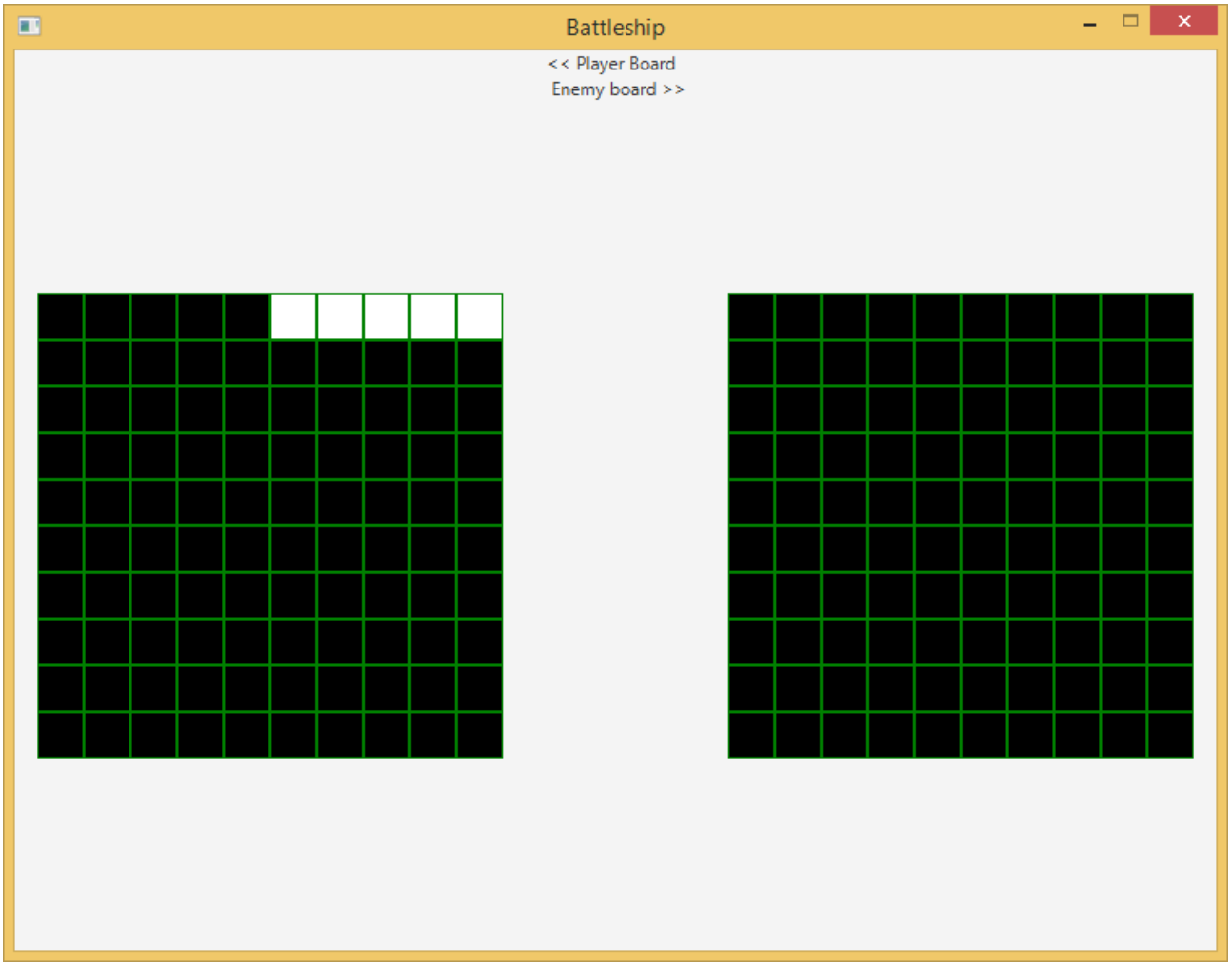
1.1 Stawianie statków

Można postawić statki w dwa sposoby:

-Lewy przycisk myszy stawia statek pionowo. Sprawdza pole, które wybraliśmy i pola poniżej czy statek może się zmieścić.



Drugim sposobem jest postawienie statku na planszy w poprzek, przy użyciu prawego przycisku myszy. Program sprawdza czy wybrane pole i pola na prawo od niego są puste i czy jest to odpowiednia długość dla danego statku.



1.2 Krótki opis działania poszczególnych funkcji i klas w projekcie.

Zacznijmy od pliku Board.java.

Metody zawarte w tym pliku prezentują się następująco:
- **public Board** – tutaj jak łatwo zauważyć po nazwie, jest to metoda która odpowiada za planszę gracza, a także planszę przeciwnika,
- **public boolean placeShip** – jest to metoda odpowiedzialna za stawianie statków na planszy,
- public Cell getCell – jest to metoda odpowiedzialna za aktualnie kliknięta komórkę,
- **private Cell[] getNeighbors** – sprawdza czy statek nie znajduje się za blisko innych statków,
- **private boolean canPlaceShip** – główna funkcja do sprawdzania, czy statek pomieści się na danym miejscu w planszy,
- **public Cell** – jest to metoda odpowiedzialna za komórki wyświetlane na ekranie na których znajdują się statki,
- **public boolean shoot()** – jest to metoda zmieniająca kolor komórki po trafieniu.

Teraz przejdźmy do skrótowego omówienia pliku Main.java

Metody zawarte w tym pliku:
- **private Parent createContent()** – tutaj mieści się cały layout naszej gry, od pozycjonowania plansz w oknie (HBox hbox1 = new HBox(playerBoard,enemyBoard);), po komunikaty związane z wygraną czy też przegraną gracza
(Alert alert1 = new Alert(AlertType.information);
alert.setTitle("End game");
alert.setHeaderText("TOU WIN");),
- **private void enemyMove()** – jest to przeciwnik w formie komputera, ale zrobiony czysto na losowości,
- **private void startGame()** – w tej metodzie odbywa się ustawianie statków przeciwnika na planszy.

Plik Ship.java zawiera podstawowe informacje o statkach typu, czy jest postawiony wertykalnie czy horyzontalnie, jaki to typ statku, czy też jego „pasek zdrowia” i to czy dalej unosi się na wodzie, czy może jest już zestrzelony.

Zawartość pliku Main.java:

```
package sample;

import java.util.Random;
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.scene.control.Label;
import sample.Board.Cell;

public class Main extends Application {

    private boolean running = false;
    private Board enemyBoard, playerBoard;

    private int shipsToPlace = 5;

    private boolean enemyTurn = false;

    private Random random = new Random();

    private Parent createContent() {
        Alert alert1 = new Alert(AlertType.information);
        alert.setTitle("End game");
        alert.setHeaderText("TOU WIN");
        alert.setContentText(null);
        BorderPane root = new BorderPane();
        root.setPrefSize(800, 600);

        Label label = new Label("<< Player Board \n Enemy board >>");

        HBox hbox3 = new HBox();
        hbox3.getChildren().add(label);
```

```
        hbox3.setAlignment(Pos.CENTER);
        root.setTop(hbox3);

        enemyBoard = new Board(true, event -> {
            if (!running)
                return;

            Cell cell = (Cell) event.getSource();
            if (cell.wasShot)
                return;

            enemyTurn = !cell.shoot();

            if (enemyBoard.ships == 1) {
                alert.showAndWait();

                System.exit(0);
            }

            if (enemyTurn)
                enemyMove();
        });

        playerBoard = new Board(false, event -> {
            if (running)
                return;

            Cell cell = (Cell) event.getSource();
            if (playerBoard.placeShip(new Ship(shipsToPlace, event.getButton()
== MouseButton.PRIMARY), cell.x, cell.y)) {

                if (--shipsToPlace == 1) {
                    startGame();
                }
            }
        });

        HBox hbox1 = new HBox(playerBoard,enemyBoard);
        hbox1.setAlignment(Pos.CENTER);
        hbox1.setSpacing(150);

        VBox vbox = new VBox();
        vbox.setAlignment(Pos.CENTER);
        vbox.getChildren().add(hbox1);

        root.setCenter(vbox);
        return root;
    }

    private void enemyMove() {
        Alert alert2 = new Alert(AlertType.information);
        alert.setTitle("End game");
        alert.setHeaderText("TOU LOSE");
        alert.setContentText(null);
        while (enemyTurn) {
            int x = random.nextInt(10);
            int y = random.nextInt(10);

            Cell cell = playerBoard.getCell(x, y);
            if (cell.wasShot)
                continue;

            enemyTurn = cell.shoot();
```

```
            if (playerBoard.ships == 0) {
                alert2.showAndWait();
                System.exit(0);
            }
        }
    }

    private void startGame() {
        // place enemy ships
        int type = 5;

        while (type > 1) {
            int x = random.nextInt(10);
            int y = random.nextInt(10);

            if (enemyBoard.placeShip(new Ship(type, Math.random() < 0.5), x, y))
{
                type--;
            }
        }

        running = true;
    }

    @Override
    public void start(Stage primaryStage) throws Exception {
        Scene scene = new Scene(createContent());
        primaryStage.setTitle("Battleship");
        primaryStage.setScene(scene);
        primaryStage.setResizable(false);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Zawartość pliku Board.java:

```
package sample;

import java.util.ArrayList;
import java.util.List;

import javafx.event.EventHandler;
import javafx.geometry.Point2D;
import javafx.scene.Parent;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;

public class Board extends Parent {
    private VBox rows = new VBox();
    private boolean enemy = false;
    public int ships = 5;

    public Board(boolean enemy, EventHandler<? super MouseEvent> handler) {
        this.enemy = enemy;
        for (int y = 0; y < 10; y++) {
```

```
            HBox row = new HBox();
            for (int x = 0; x < 10; x++) {
                Cell c = new Cell(x, y, this);
                c.setOnMouseClicked(handler);
                row.getChildren().add(c);
            }

            rows.getChildren().add(row);
        }

        getChildren().add(rows);
    }

    public boolean placeShip(Ship ship, int x, int y) {
        if (canPlaceShip(ship, x, y))
        {
            int length = ship.type;

            if (ship.vertical)
            {
                for (int i = y; i < y + length; i++)
                {
                    Cell cell = getCell(x, i);
                    cell.ship = ship;
                    if (!enemy)
                    {
                        cell.setFill(Color.CORNFLOWERBLUE);
                        cell.setStroke(Color.GREEN);
                    }
                }
            }
            else
                {
                    for (int i = x; i < x + length; i++)
                    {
                        Cell cell = getCell(i, y);
                        cell.ship = ship;
                        if (!enemy)
                        {
                            cell.setFill(Color.CORNFLOWERBLUE);
                           cell.setStroke(Color.GREEN);
                        }
                    }
                }
            return true;
        }
        return false;
    }

    /*
    * Getting the cell that was clicked on.*
    * */
    public Cell getCell(int x, int y) {
        return (Cell)((HBox)rows.getChildren().get(y)).getChildren().get(x);
    }

    /*
    * Checking if ship can be placed and isn't close to other ship.
    * */
    private Cell[] getNeighbors(int x, int y) {
        Point2D[] points = new Point2D[] {
                new Point2D(x - 1, y),
                new Point2D(x + 1, y),
                new Point2D(x, y - 1),
```

```
                new Point2D(x, y + 1)
        };

        List<Cell> neighbors = new ArrayList<Cell>();

        for (Point2D p : points) {
            if (isValidPoint(p)) {
                neighbors.add(getCell((int)p.getX(), (int)p.getY()));
            }
        }

        return neighbors.toArray(new Cell[0]);
    }

    /*
    * Main check for placing ship.
    * */
    private boolean canPlaceShip(Ship ship, int x, int y) {
        int length = ship.type;
        /*
        * Checking spaces vertically
        * */
        if (ship.vertical) {
            for (int i = y; i < y + length; i++) {
                if (!isValidPoint(x, i))
                    return false;

                Cell cell = getCell(x, i);
                if (cell.ship != null)
                    return false;

                for (Cell neighbor : getNeighbors(x, i)) {
                    if (!isValidPoint(x, i))
                        return false;

                    if (neighbor.ship != null)
                        return false;
                }
            }
        }
        else
            { //Checking spaces horizontally.
                for (int i = x; i < x + length; i++)
                {
                    if (!isValidPoint(i, y))
                        return false;

                    Cell cell = getCell(i, y);
                    if (cell.ship != null)
                        return false;

                    for (Cell neighbor : getNeighbors(i, y))
                    {
                        if (!isValidPoint(i, y))
                            return false;

                        if (neighbor.ship != null)
                            return false;
                    }
                }
            }

        return true;
    }
```

```
    private boolean isValidPoint(Point2D point) {
        return isValidPoint(point.getX(), point.getY());
    }

    private boolean isValidPoint(double x, double y) {
        return x >= 0 && x < 10 && y >= 0 && y < 10;
    }

    public class Cell extends Rectangle {
        public int x, y;
        public Ship ship = null;
        public boolean wasShot = false;

        private Board board;

        /*
        * Size of tiles
        * */
        public Cell(int x, int y, Board board) {
            super(30, 30);
            this.x = x;
            this.y = y;
            this.board = board;
            setFill(Color.BLACK);
            setStroke(Color.GREEN);
        }

        /*
        * Changing color if shot.
        * 2nd if changes color to red if ship was on that tile.
        * */

        public boolean shoot() {
            wasShot = true;
            setFill(Color.WHITE);

            if (ship != null) {
                ship.hit();
                setFill(Color.RED);
                if (!ship.isAlive()) {
                    board.ships--;
                }
                return true;
            }

            return false;
        }
    }
}
```

Zawartość pliku Ship.java

```
package sample;

import javafx.scene.Parent;

public class Ship extends Parent {
    public int type;
    public boolean vertical = true;
    private int health;
```

```
    public Ship(int type, boolean vertical) {
        this.type = type;
        this.vertical = vertical;
        health = type;
    }

    public void hit() {
        health--;
    }

    public boolean isAlive() {
        return health > 0;
    }
}
```

1.3 Krótka lista zasad.

W przeciwieństwie do standardowej wersji statków, jeśli trafi się w statek przeciwnika tura nie przechodzi na przeciwnika lecz można kontynuować ostrzał, działa w obie strony.

Nie można grupować statków jeden obok drugiego. Aplikacja nie pozwoli postawić statków stykających się bokami. Musi istnieć między nimi 1 kratka przerwy. Nie sprawdzane jest jednak na ukos.