Radosław Kopeć 305333
Projektowanie obiektowe
Laboratorium 3
Wzorce projektowe
Kwiecień 2020

4.1 Builder

1. Tworzymy interfejs MazeBuilder służący do tworzenia labiryntów.
Muszą w nim być zawarte motody do tworzenia elementów labiryntu.

```
public interface MazeBuilder {

    void reset();
    void addRoom(Room room);
    void setDoorBetweenRooms(Room room1,Room room2);
    void setWallBetweenRooms(Direction firstRoomDirection,Room room1,Room room2);

}
```

Labirynt składa się z pomieszczeń, które możemy łączyć drzwiami lub ścianami.
Zakładam, że aby stworzyć drzwi najpierw pokoje muszą mieć wspólną ścianę, dlatego przy tworzeniu drzwi nie podaję kierunku.

2.Po utworzeniu powyższego interfejsu modyfikujemy funkcje składową tak, aby przyjmowała jako parametr obiekt tej klasy

```
public void createMaze(MazeBuilder mazeBuilder){ ...
```

3. Prześledź i zinterpretuj co dały obecne zmiany (krótko opisz swoje spostrzeżenia).

Zmiany te pozwoliły wyodrębnić tworzenie złożonego obiektu do innej klasy. Pozwala to uniknąć dużych złożonych konstruktorów z wieloma parametrami, oraz wprowadza to zasadę pojedynczej odpowiedzialności. Będziemy mogli tworzyć labirynt krok po kroku. Dzięki interfejsowi będziemy mogli kilka razy użyć tego samego kodu do budowania różnych reprezentacji produktu.

4. Stwórz klasę StandardBuilderMaze będącą implementacją MazeBuildera. Powinna ona mieć zmienną currentMaze, w której jest zapisywany obecny stan labiryntu. Powinniśmy móc: tworzyć pomieszczenie i ściany w okół niego, tworzyć drzwi pomiędzy pomieszczeniami (czyli musimy wyszukać odpowiednie pokoje oraz ścianę, która je łączy). Dodaj tam dodatkowo metodę prywatną CommonWall, która określi kierunek standardowej ściany pomiędzy dwoma pomieszczeniami.

StandardBuilderMaze:

Metody reset i getResult pozwalają tworzyć nowy obiekt za pomocą buildera lub zwrócić obecną instancje tworzonego obiektu. currentMaze przechowuje aktualny stan labiryntu.

```
public class StandardBuilderMaze implements MazeBuilder {
```

```
    private Maze currentMaze = new Maze();
    public Maze getResult(){
        return this.currentMaze;
    }

    @Override
    public void reset(){
        this.currentMaze = new Maze();
    }
```

Tworząc pokój od razu tworzymy 4 ściany dla niego, w późniejszych funkcjach będziemy mogli zamienić te ściany na drzwi lub połączyć je z innym pokojem.

```
    @Override
    public void addRoom(Room room) {
        room.setSide(Direction.North, new Wall());
        room.setSide(Direction.East, new Wall());
        room.setSide(Direction.South, new Wall());
        room.setSide(Direction.West, new Wall());
        this.currentMaze.addRoom(room);
    }
```

Za pomocą tej funkcji tworzymy drzwi między pokojami.

```
    @Override
    public void setDoorBetweenRooms(Room room1,Room room2){
        Door door = new Door(room1,room2);
        Direction dir = commonWall(room1,room2);
        if(dir == null){
            System.out.println("Pomieszczenia nie mają wspólnej sciany");
            return;
        }
        room1.setSide(dir,door);
        room2.setSide(Direction.getOpposite(dir),door);
    }
```

Funkcja commonWall zwraca kierunek ściany pierwszego pokoju, który jest wspólny z drugim pokojem. Porównuje przeciwne kierunki ścian ponieważ w standardowym labiryncie bez magicznych własności PÓŁNOCNA ściana pokoju 1 jest POŁUDNIOWĄ ŚCIANĄ POKOJU 2 (jeśli pokój 2 jest nad pokojem 1)

```
    private Direction commonWall(Room room1,Room room2){
        for (Direction dir : Direction.values()){
            if(room1.getSide(dir) == room2.getSide(Direction.getOpposite(dir))){
                if(room1.getSide(dir) != null){
                    return dir;
                }
```

```
            }
        }
        return null;
    }
```

Funckja setWallBetweenRooms łączy pokoje za pomocą ściany według kierunku ściany pierwszego pokoju, oznacza to że ściana PÓŁNOCNA pokoju 1 staje się również ścianą POŁUDNIOWĄ pokoju 2 itp.

```
    @Override
    public void setWallBetweenRooms(Direction firstRoomDirection, Room room1, Room room2) {
        if(room1.getSide(firstRoomDirection) == null){
            room1.setSide(firstRoomDirection,new Wall());
        }
room2.setSide(Direction.getOpposite(firstRoomDirection),room1.getSide(firstRoomDirection));
    }
```

Direction

Do Direction dodajemy metodę zwracającą przeciwny kierunek.

```
    public static Direction getOpposite(Direction d){
        switch (d){
            case East:
                return West;
            case West:
                return East;
            case North:
                return South;
            case South:
                return North;
        }
        return null;
    }
```

5. Tworzymy labirynt przy pomocy operacji createMaze, gdzie parametrem będzie obiekt klasy StandardMazeBuilder.

Main

Tworzymy nowego StandardBuilderMaze, przekazujemy go w parametrze funkcji creatMaze i pobieramy result. Taka konstrukcja pozwala nam wywoływać konstruowanie produktów dla różnych Builderów za pomocą tego samego kodu, w ten sam sposób.

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

        MazeGame mazeGame = new MazeGame();
        StandardBuilderMaze standardBuilderMaze = new StandardBuilderMaze();
        mazeGame.createMaze(standardBuilderMaze);
        Maze maze = standardBuilderMaze.getResult();
```

MazeGame

Tworzymy prosty labirynt z 4 pokojami:

```
    public void createMaze(MazeBuilder mazeBuilder){
        Room r1 = new Room(1);
        Room r2 = new Room(2);
        Room r3 = new Room(3);
        Room r4 = new Room(4);

        mazeBuilder.addRoom(r1);
        mazeBuilder.addRoom(r2);
        mazeBuilder.addRoom(r3);
        mazeBuilder.addRoom(r4);

        mazeBuilder.setWallBetweenRooms(Direction.South,r1,r2);
        mazeBuilder.setWallBetweenRooms(Direction.East,r2,r3);
        mazeBuilder.setWallBetweenRooms(Direction.North,r3,r4);
        mazeBuilder.setWallBetweenRooms(Direction.East,r1,r4);

        mazeBuilder.setDoorBetweenRooms(r1,r2);
        mazeBuilder.setDoorBetweenRooms(r2,r3);
        mazeBuilder.setDoorBetweenRooms(r3,r4);

        //Maze like:
//         ___
//        | | |   1 4
//        |___|   2 3
//
    }
```

6. Tworzymy kolejną podklasę MazeBuildera o nazwie CountingMazeBuilder. Budowniczy tego obiektu w ogóle nie tworzy labiryntu, a jedynie zlicza utworzone komponenty różnych rodzajów. Powinien mieć metodę GetCounts, która zwraca ilość elementów.

Tworzymy nową klasę CountingMaze reprezentującą ilość elementów w labiryncie:

CountingMaze

```
public class CountingMaze {

    private  int walls;
    private int rooms;
    private int doors;

    CountingMaze(){
        this.doors = 0;
```

```
        this.walls = 0;
        this.rooms = 0;
    }

//niżej gettery i settery
```

Tworzymy Buildera dla tej klasy:

CountingMazeBuilder

Funkcje poboru i resetowania obecnego obiektu klasy CountingMaze.

```
public class CountingMazeBuilder implements MazeBuilder {

    CountingMaze countingMaze = new CountingMaze();

    @Override
    public void reset() {
        this.countingMaze = new CountingMaze();
    }

    public CountingMaze getResult(){
        return this.countingMaze;
    }
```

Gdy dodajemy pokoj zwiększamy licznik pokoju o 1, ale również licznik ścian o 4 bo nasz pokój na starcie ma 4 sciany:

```
    @Override
    public void addRoom(Room room) {
        this.countingMaze.setRooms(this.countingMaze.getRooms() + 1);
        this.countingMaze.setWalls(this.countingMaze.getWalls() + 4);
    }
```

Gdy dodajemy Drzwi to zastepujemy jedną sciane drzwiami więc odejmujemy jedną ścianę i dodajemy jedne drzwi

```
    @Override
    public void setDoorBetweenRooms(Room room1, Room room2) {
        this.countingMaze.setDoors(this.countingMaze.getDoors() +1);
        this.countingMaze.setWalls(this.countingMaze.getWalls() - 1);
    }
```

Gdy dodajemy ścianę to tak naprawdę nadpisujemy jedną ścianę z room2 jedną ścianą z room1, więc odejmujemy jedną śćianę z licznika:

```
   @Override
    public void setWallBetweenRooms(Direction firstRoomDirection, Room room1, Room room2) {
        this.countingMaze.setWalls(this.countingMaze.getWalls() - 1);
    }
```

Aby pokazac ze działa w mainie:

```
        CountingMazeBuilder countingMazeBuilder = new CountingMazeBuilder();
        mazeGame.createMaze(countingMazeBuilder);
        CountingMaze countingMaze = countingMazeBuilder.getResult();

        System.out.println("Ściany : ");
        System.out.println(countingMaze.getWalls());
        System.out.println("Drzwi : ");
        System.out.println(countingMaze.getDoors());
        System.out.println("Pokoje : ");
        System.out.println(countingMaze.getRooms());
```

Tak jak oczekiwaliśmy, wynik: dla takiego labiryntu

```
//         ____
//        | | |
//        |___|
```

```
Ściany :
9
Drzwi :
3
Pokoje :
4
```

mamy 4 pokoje (4 kwadraciki) 3 drzwi (3 puste sciany miedzy pokojami), 9 scian (9 kresek)

4.2 Fabryka Abstrakcyjna

1. Tworzymy klasę MazeFactory, która służy do tworzenia elementów labiryntu. Można jej użyć w programie, który np. wczytuje labirynt z pliku .txt , czy generuje labirynt w sposób losowy.

```
public interface MazeFactory {

    Room createRoom(int n);
    Wall createWall();
    Door createDoor(Room r1, Room r2);
}
```

2. Przeprowadzamy kolejną modyfikacje funkcji createMaze tak, aby jako parametr brała MazeFactory.

```
public class MazeGame {

    public void createMaze(MazeBuilder mazeBuilder, MazeFactory factory){
```

3. Tworzymy klasę EnchantedMazeFactory (fabryka magicznych labiryntów), która dziedziczy z MazeFactory. Powinna przesłaniać kilka funkcji składowych i zwracać różne podklasy klas Room, Wall itd. (należy takie klasy również stworzyć).

Tworzymy EnchantedDoor,EnchantedWall, oraz EnchantedRoom dziedziczące odpowiednio po Door, Wall, oraz Room.

```
public class EnchantedDoor extends Door {

    public EnchantedDoor(Room r1, Room r2) {
        super(r1, r2);
    }

    @Override
    public void Enter(){
        System.out.println("Wchodzisz przez magiczne drzwi");
    }

}
```

```
public class EnchantedRoom extends Room {

    public EnchantedRoom(int number) {
        super(number);
    }

    @Override
    public void Enter(){
        System.out.println("To jest magiczny pokoj");
    }
}
```

```
public class EnchantedWall extends Wall {

    @Override
    public void Enter(){
        System.out.println("Magiczna sciana");
    }
}
```

Nastepnie tworzymy klase EnchantedMazeFactory

```
public class EnchantedMazeFactory implements MazeFactory{

    @Override
    public Room createRoom(int number) {
        return new EnchantedRoom(number);
}

    @Override
    public Wall createWall() {
        return new EnchantedWall();
    }

    @Override
    public Door createDoor(Room r1,Room r2) {
        return new EnchantedDoor(r1,r2);
    }
}
```

4. Stwórz klasę BombedMazeFactory, która zapewnia, że ściany to obiekty klasy BombedWall, a pomieszczenia to obiekty klasy BombedRoom.

Tworzymy BombedDoor,BombedWall, oraz BombedRoom dziedziczące odpowiednio po Door, Wall, oraz Room.

```
public class BombedDoor extends Door {
    public BombedDoor(Room r1, Room r2) {
        super(r1, r2);
    }

    @Override
    public void Enter(){
        System.out.println("Wchodzisz przez wybuchowe drzwi");
    }
}
```

```
public class BombedWall extends Wall {

    @Override
    public void Enter(){
        System.out.println("Wybuchowa sciana");
    }
}
```

```
public class BombedRoom extends Room {

    public BombedRoom(int number) {
```

```
        super(number);
    }

    @Override
    public void Enter(){
        System.out.println("Masz 15s na podejcie decyzji!");
    }
}
```

Tworzymy BombedMazeFactory implementujaca MazeFactory

```
public class BombedMazeFactory implements MazeFactory {

    @Override
    public Room createRoom(int n) {
        return new BombedRoom(n);
    }

    @Override
    public Wall createWall() {
        return new BombedWall();
    }

    @Override
    public Door createDoor(Room r1, Room r2) {
        return new BombedDoor(r1,r2);
    }
}
```

4.3 Singleton

Wprowadzamy w powyżej stworzonej implementacji mechanizm, w którym MazeFactory będzie Singletonem. Powinien być on dostępny z pozycji kodu, który jest odpowiedzialny z tworzenie poszczególnych części labiryntu.

Problem
Jako, że zaimplementowałem MazeFactory jako interface to mechanizm Singletona zaimplementuje, w każdej z klas ConctereMazeFactory. Pozwoli mi to mieć pojedynczą instancję każdej z Fabryk, ale nie jedną fabrykę (np tylko EnchantedMazeFactory), musiałbym zamienić MazeFactory na klasę (nie interface), imeplementowała by ona mechanizm singletona, ale miałoby to jedną wadę nie da się wymusić tworzenia prywatnych konstruktorów na podklasach.

EnchantedMazeFactory

```
public final class EnchantedMazeFactory implements MazeFactory{
```

```
    private static EnchantedMazeFactory instance;

    // slow initialization
    private EnchantedMazeFactory(){
        try {
            Thread.sleep(1000);
        }catch (InterruptedException ex){
            ex.printStackTrace();
        }
    }

    public static EnchantedMazeFactory getInstance(){
        if (instance == null){
            instance = new EnchantedMazeFactory();
        }
        return instance;
    }
```

BombedMazeFactory

```
public final class BombedMazeFactory implements MazeFactory {

    private static BombedMazeFactory instance;

    private BombedMazeFactory(){
        try{
            Thread.sleep(1000);
        }catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }

    public static BombedMazeFactory getInstance(){
        if (instance == null){
            instance = new BombedMazeFactory();
        }
        return  instance;
    }
```

Labirynt tworzymy w jednym z builderów, musimy tam wprowadzić możliwość kontroli jaki rodzaj ścian budujemy. Musimy zatem do StandadBuilderMaze przekazywać fabrykę, by za jej pomocą tworzył odpowiednie obiekty.

Do StandatBuilderMaze dodajemy zatem odpowiedni konstruktor i pole:

```
    private MazeFactory factory;
```

```
    public StandardBuilderMaze(MazeFactory factory){
        this.factory = factory;
    }
```

oraz modyfikujemy metody wszędzie tam gdzie są tworzone nowe obiekty labiryntu.

```
    public void addRoom(Room room) {
        room.setSide(Direction.North, factory.createWall());
        room.setSide(Direction.East, factory.createWall());
        room.setSide(Direction.South, factory.createWall());
        room.setSide(Direction.West, factory.createWall());
        this.currentMaze.addRoom(room);
    }

     public void setDoorBetweenRooms(Room room1,Room room2){
        Door door = factory.createDoor(room1,room2);

    public void setWallBetweenRooms(Direction firstRoomDirection, Room room1, Room room2) {
        if(room1.getSide(firstRoomDirection) == null){
            room1.setSide(firstRoomDirection,factory.createWall());
        }
```

4.4 Rozszerzenie aplikacji labirynt

b) Zademonstruj, że MazeFactory faktycznie jest Singletonem (najłatwiej stworzyć przykład, w którym się sprawdza, czy obiekt zwracany przy 2 konstrukcji to faktycznie ten sam, który został stworzony na początku).
Aby sprawdzić, że fabryka rzeczywiśćie jest Singletonem wprowadzamy następujący kod do maina i kompilujemy go.

```
        MazeFactory factory = EnchantedMazeFactory.getInstance();
        MazeFactory factory2 = EnchantedMazeFactory.getInstance();
        if(factory == factory2){
            System.out.println("To te same obiekty!");
        }
        else{
            System.out.println("Mamy roznie obiekty");
        }
```

Wynik

```
"C:\Program Files\Java\jd
To te same obiekty!
```

Jak widać zaimplementowany mechanizm Singletona działa.

a) Zrezygnowałem z prostego wyświetlania na rzecz podpunktu **4.5** i od razu zaimplementowałem wyświetlanie w JavieFX

4.5 Dla chętnych!

Moim zdaniem stworzenie clasy MazeGame jako Singleton miałoby sens, tworzyłoby to wtedy możliwość dostępu do gry z poziomu metody Enter() w każdym obiekcie co umożliwiłoby integracje. Na przykład BombedRoom mogłoby uśmiercić gracza po 15sekundach.

Modyfikacja **MazeGame**, aby była Singletonem i przechowywała gracza, oraz komunikat do wyświetlenia w panelu.

```
public class MazeGame {

    private static MazeGame instance;
    private Player player;
    private String communicat = "";

    public String getCommunicat() {
        return communicat;
    }

    private MazeGame(){
        try{
            Thread.sleep(1000);
        }catch (InterruptedException ex){
            ex.printStackTrace();
        }
    }

    public static MazeGame getInstance(){
        if(instance == null){
            instance = new MazeGame();
        }
        return instance;
    }
```

Dodanie funkcji pozwalajacyn na powiazanie pomiedzy obiekatami labiryntu a grą, graczem:

```
 public void movePlayerLeft(){
        player.moveLeft();
    }
    public void movePlayerRgiht(){
        player.moveRigth();
    }
    public void movePlayerAhead(){
        player.goAhead();
```

```
    }
    public void changePlayerRoom(Room r1,Room r2){
        if(player.getRoom() == r1) {
            player.setRoom(r2);
        }
        else {
            player.setRoom(r1);
        }
    }
    public void changePlayerHealth(int i){
        player.setHealth(player.getHealth() + i);
        System.out.println(player.getHealth());
    }

    public void setCommunicat(String s){
        communicat = s;
    }
```

Wprowadzenie głównej pętli gry:

```
public void startGame(Maze maze){

        player = new Player(Direction.South,maze.getStartRooom(),20);
        MazeViewer viewer = new MazeViewer(maze,player);
        GameFrame frame = new GameFrame(maze,player);
        boolean exit = true;
        while(exit){

            try {
                Thread.sleep(50);
            }catch (InterruptedException ex){
                ex.printStackTrace();
            }

            frame.paint();

            if(player.getHealth() <= 0){
                communicat =  "Zginales z powodu obrazen";
                break;
            }
            if(player.getRoom() == maze.getEndRoom()){
                communicat =  "Jests na mecie gratulacje!";

            }
            if(player.getRoom() == maze.getStartRooom()){
                communicat =  "Jestes na starcie";

            }
```

```
        }

    }
```

Dodanie klasy Player reprezentującej gracza:

```
public class Player {

    private Room room;
    private Direction direction;
    private int health;

    public int getHealth() {
        return health;
    }

    public void setHealth(int health) {
        this.health = health;
    }


    public Player(Direction startDirection, Room startRoom,int startHealth){
        room = startRoom;
        direction = startDirection;
        health = startHealth;
    }

    public void goAhead(){
        room.getSide(direction).Enter();
    }

    public void moveLeft(){
        direction = Direction.getNext(direction);
    }
    public void moveRigth(){
        direction = Direction.getBefore(direction);
    }

    public Direction getDirection() {
        return direction;
    }

    public void setDirection(Direction direction) {
        this.direction = direction;
    }
    public Room getRoom() {
        return room;
    }

    public void setRoom(Room room) {
```

```
        this.room = room;
    }

}
```

Modyfikacja funkcji Enter w komponentach Labiryntu, pozwala nam zdefiniować indywidualne zachowanie dla różnych rodzin klas, różne dla Enchanted i Bombed.

EnchantedWall

```
    @Override
    public void Enter(){
        wallStamina--;
        MazeGame.getInstance().setCommunicat("Uderzyles w magiczna sciane, zaczynasz
krwawić");

        if(wallStamina < 0){
            MazeGame.getInstance().setCommunicat("Orzymano obrazenia");
            MazeGame.getInstance().changePlayerHealth(-10);
        }
    }
```

EnchantedDoor

```
    @Override
    public void Enter(){
        MazeGame.getInstance().setCommunicat("Przeszedles przez magiczne drzwi!");
        MazeGame.getInstance().changePlayerRoom(getRoom1(),getRoom2());
    }
```

Dodanie klas obsługujących wyświetlanie, aplikację okienkową:

GameFrame

Obsługa klawiatury:

```
 @Override
    public void keyPressed(KeyEvent e) {
        int i = e.getKeyCode();

        if (i == KeyEvent.VK_A || i == KeyEvent.VK_LEFT) {
            MazeGame.getInstance().movePlayerLeft();
        }

        if (i == KeyEvent.VK_D || i == KeyEvent.VK_RIGHT) {
            MazeGame.getInstance().movePlayerRgiht();
```

```
        }
        if (i == KeyEvent.VK_W || i == KeyEvent.VK_UP) {
            MazeGame.getInstance().movePlayerAhead();
        }
    }
```

Funkcje inicjalizujące okno oraz, funkcja paint renderująca panel ponownie:

```
public class GameFrame extends JFrame implements ActionListener,KeyListener {

    JPanel panel;
    public GameFrame(Maze maze, Player player) {
         super("Maze");
         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         this.setSize(600,600);
         setLocationRelativeTo(null);

         panel = new GamePanel(maze,player,this);
         add(panel);

         addKeyListener(this);
         setVisible(true);
    }

    public void paint(){
        panel.repaint();
    }
```

GamePanel

Obsługuje wyświetlanie labiryntu, pozycji gracza, oraz komunikatów

Funkcja drawRoom, rysująca pokoj i jego komponenty (sciany, drzwi)

```
public void drawRoom(int weight,int height,Graphics g,Room room){

        g.setColor(Color.WHITE);
        if(maze.getStartRooom() == room)
            g.setColor(Color.BLUE);
        if(maze.getEndRoom() == room)
            g.setColor(Color.GREEN);

        g.fillRect(weight,height,SCALE,SCALE);

        for(Direction d: Direction.values()){

            if(room.getSide(d) instanceof Wall){
                g.setColor(Color.black);
            }
```

```
            else if(room.getSide(d) instanceof Door){
                g.setColor(Color.WHITE);
            }

            switch (d){
                case West:
                    g.fillRect(weight ,height+BOLD , BOLD , SCALE -BOLD);
                    break;
                case East:
                    g.fillRect(weight + SCALE -BOLD,height + BOLD, BOLD , SCALE - BOLD );
                    break;
                case North:
                    g.fillRect(weight,height, SCALE , BOLD);
                    break;
                case South:
                    g.fillRect(weight  ,height + SCALE - BOLD, SCALE ,  BOLD);
                    break;

            }

            if(player.getRoom() == room){
                g.setColor(Color.RED);
                switch (player.getDirection()){
                    case West:
                        int[] tabx = {weight + 2*BOLD,weight + SCALE - 2*BOLD,weight + SCALE
- 3*BOLD, weight + SCALE - 2*BOLD};
                        int[] taby = {height + SCALE/2,height + 2*BOLD,height +
SCALE/2,height + SCALE - 2*BOLD};
                        g.fillPolygon(tabx,taby,4);
                        break;
                    case East:
                        int[] tabx2 = {weight + SCALE - 2*BOLD,weight + 2*BOLD,weight +
3*BOLD, weight + 2*BOLD};
                        int[] taby2 = {height + SCALE/2,height + 2*BOLD,height +
SCALE/2,height + SCALE - 2*BOLD};
                        g.fillPolygon(tabx2,taby2,4);
                        break;
                    case North:
                        int[] taby3 = {height + 2*BOLD,height + SCALE - 2*BOLD,height + SCALE
- 3*BOLD, height + SCALE - 2*BOLD};
                        int[] tabx3 = {weight + SCALE/2,weight + 2*BOLD,weight +
SCALE/2,weight + SCALE - 2*BOLD};
                        g.fillPolygon(tabx3,taby3,4);
                        break;
                    case South:
                        int[] taby4 = {height + SCALE - 2*BOLD,height + 2*BOLD,height +
3*BOLD, height + 2*BOLD};
                        int[] tabx4 = {weight + SCALE/2,weight + 2*BOLD,weight +
SCALE/2,weight + SCALE - 2*BOLD};
                        g.fillPolygon(tabx4,taby4,4);
```

```
                    break;

                }
            }
        }

    }
```

Funkcja drawMazeBfs za pomocą BFS-a przechodzi po labiryncie i rysuje powiązane ze sobą pokoje, pozwala to na tworzenie labiryntu o dowolnym kształcie korytarzy:

```
public void  drawMazeBfs(Graphics g,int x,int y,Room r,List<Integer> roomsColors){
        if(roomsColors.contains(r.getRoomNumber()))return;
        drawRoom(x,y,g,r);
        roomsColors.add(r.getRoomNumber());
        for(Direction d: Direction.values()){
            for (Room r2 : maze.getRooms()) {
                if (r.getSide(d) == r2.getSide(Direction.getOpposite(d))) {
                    switch (d) {
                        case West:
                            drawMazeBfs(g, x - SCALE,y, r2,roomsColors);
                            break;
                        case East:
                            drawMazeBfs(g, x + SCALE,y,  r2,roomsColors);
                            break;
                        case North:
                            drawMazeBfs(g, x ,y - SCALE,  r2,roomsColors);
                            break;
                        case South:
                            drawMazeBfs(g, x ,y  + SCALE,  r2,roomsColors);
                            break;
                    }
                }
            }
        }
    }
```

Funkcja paintComponent przemalowywująca obraz, konstruktor Panelu, oraz stałe potrzebne do rysowania.

```
public class GamePanel extends JPanel {

    Maze maze;
    Player player;
    int WIDTH;
    int HEIGHT;
    int SCALE = 50;
    int start_x;
    int stary_y;
```

```
    int BOLD = 6;
    GameFrame gameFrame;
    public GamePanel(Maze maze,Player player,GameFrame gameFrame) {
        this.gameFrame = gameFrame;
        this.player = player;
        this.maze = maze;
        this.HEIGHT = gameFrame.getHeight();
        this.WIDTH = gameFrame.getWidth();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        this.setSize(gameFrame.getWidth(),gameFrame.getHeight()-38);
        this.setLocation(0, 0);
        start_x = 0;
        stary_y = 30;

        int length = maze.getRooms().size();
        List<Integer> roomsColors = new ArrayList<>(length);
        drawMazeBfs(g,start_x,stary_y,maze.getStartRooom(),roomsColors);

        g.setColor(Color.BLACK);
        g.drawString(MazeGame.getInstance().getCommunicat(),20,20);
    }
```

Main

W mainie wybieramy, rodzaj komponentu labiryntów (poprzez wybór Fabryki), oraz buildera budującego standardowy labirynt.

```
public class Main {

    public static void main(String[] args) {

        MazeGame mazeGame = MazeGame.getInstance();
        MazeFactory factory = EnchantedMazeFactory.getInstance();
        StandardBuilderMaze standardBuilderMaze = new StandardBuilderMaze(factory);
        mazeGame.createMaze(standardBuilderMaze,factory);
        Maze maze = standardBuilderMaze.getResult();

        mazeGame.startGame(maze);

    }
}
```

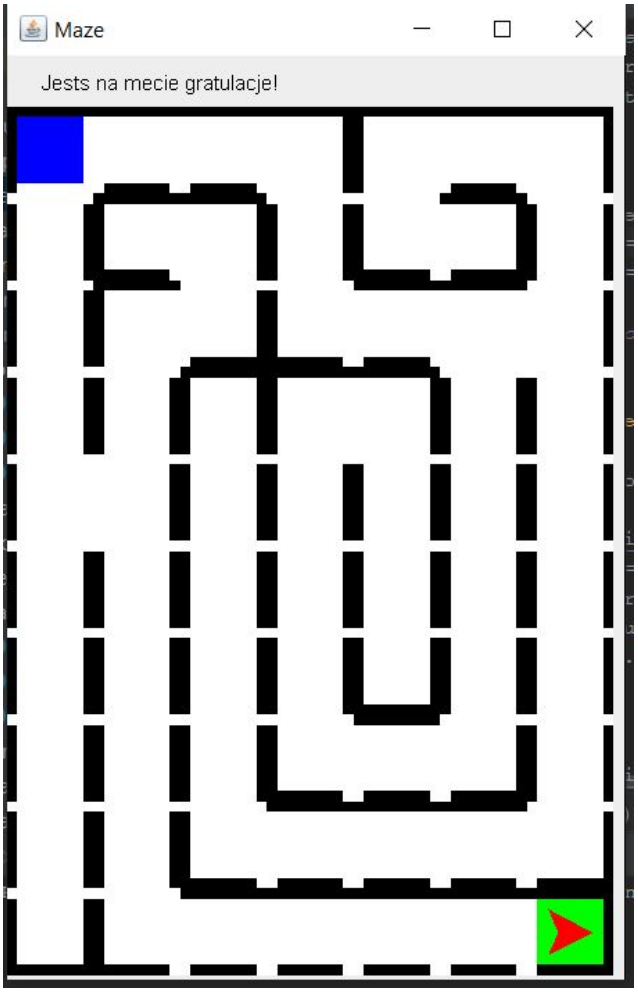
Zmodyfikowana została również funkcja createMaze w MazeGame tak aby tworzyła poniższy labirynt za pomocą buildera:

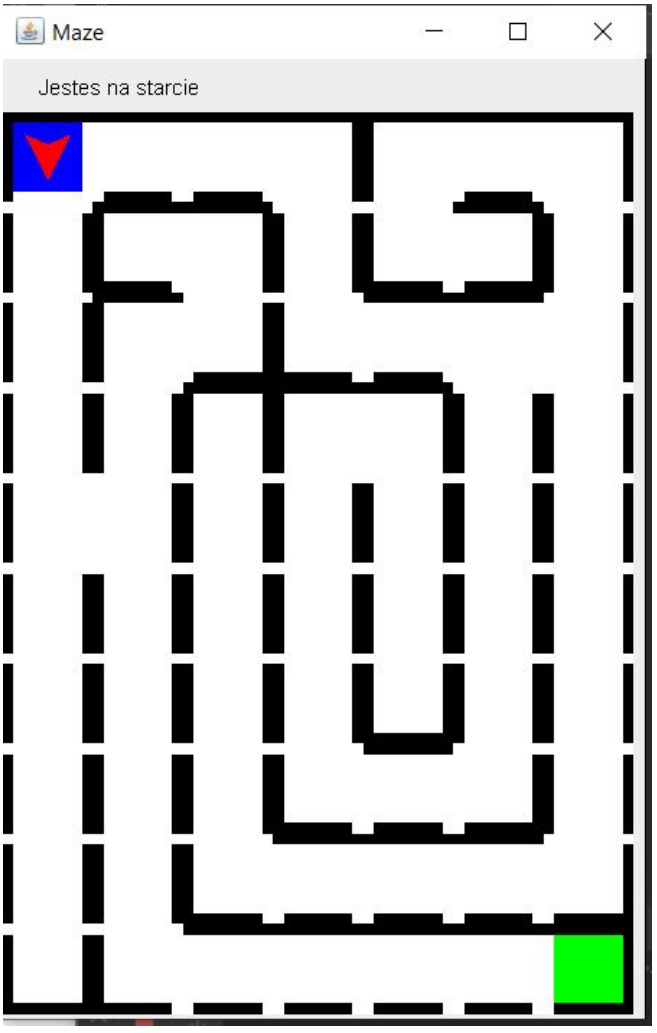
```
 public void createMaze(MazeBuilder mazeBuilder, MazeFactory factory){

        List<Room> rooms = new ArrayList<>(100);

        for(int i =0;i<7;i++){
            for(int j=0;j<10;j++){
                Room r = factory.createRoom(10*i+j);
                mazeBuilder.addRoom(r);
                rooms.add(r);
            }
        }

        for(int i =0;i<6;i++){
            for(int j=0;j<10;j++){
                mazeBuilder.setWallBetweenRooms(Direction.East,rooms.get(10*i +
j),rooms.get(10*(i+1)+j));
            }
        }

        for(int i =0;i<7;i++){
            for(int j=0;j<9;j++){
                mazeBuilder.setWallBetweenRooms(Direction.South,rooms.get(10*i +
j),rooms.get(10*(i)+j+1));
            }
        }

        mazeBuilder.createStartRoom(rooms.get(0));
        mazeBuilder.createEndRoom(rooms.get(69));

        mazeBuilder.setDoorBetweenRooms(rooms.get(0),rooms.get(1));
        mazeBuilder.setDoorBetweenRooms(rooms.get(1),rooms.get(2));
        mazeBuilder.setDoorBetweenRooms(rooms.get(2),rooms.get(3));
        mazeBuilder.setDoorBetweenRooms(rooms.get(3),rooms.get(4));
        mazeBuilder.setDoorBetweenRooms(rooms.get(4),rooms.get(5));
        mazeBuilder.setDoorBetweenRooms(rooms.get(5),rooms.get(6));
        mazeBuilder.setDoorBetweenRooms(rooms.get(6),rooms.get(7));
        mazeBuilder.setDoorBetweenRooms(rooms.get(7),rooms.get(8));
        mazeBuilder.setDoorBetweenRooms(rooms.get(8),rooms.get(9));
        mazeBuilder.setDoorBetweenRooms(rooms.get(0),rooms.get(10));
        mazeBuilder.setDoorBetweenRooms(rooms.get(10),rooms.get(20));
        mazeBuilder.setDoorBetweenRooms(rooms.get(20),rooms.get(30));
        mazeBuilder.setDoorBetweenRooms(rooms.get(30),rooms.get(31));
        mazeBuilder.setDoorBetweenRooms(rooms.get(31),rooms.get(32));
        mazeBuilder.setDoorBetweenRooms(rooms.get(32),rooms.get(42));
        mazeBuilder.setDoorBetweenRooms(rooms.get(42),rooms.get(52));

        mazeBuilder.setDoorBetweenRooms(rooms.get(52),rooms.get(62));
        mazeBuilder.setDoorBetweenRooms(rooms.get(62),rooms.get(61));
        mazeBuilder.setDoorBetweenRooms(rooms.get(61),rooms.get(60));
        mazeBuilder.setDoorBetweenRooms(rooms.get(60),rooms.get(50));
```

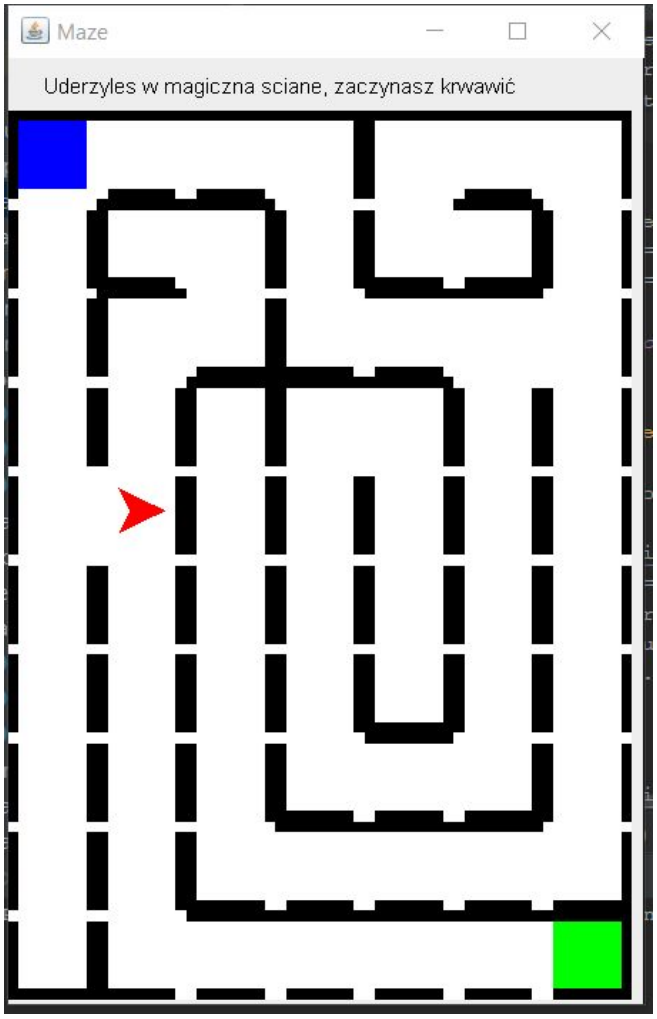
```
        mazeBuilder.setDoorBetweenRooms(rooms.get(50),rooms.get(40));
        mazeBuilder.setDoorBetweenRooms(rooms.get(40),rooms.get(41));
        mazeBuilder.setDoorBetweenRooms(rooms.get(41),rooms.get(51));
        mazeBuilder.setDoorBetweenRooms(rooms.get(62),rooms.get(63));
        mazeBuilder.setDoorBetweenRooms(rooms.get(63),rooms.get(64));
        mazeBuilder.setDoorBetweenRooms(rooms.get(64),rooms.get(65));
        mazeBuilder.setDoorBetweenRooms(rooms.get(65),rooms.get(66));
        mazeBuilder.setDoorBetweenRooms(rooms.get(20),rooms.get(30));
        mazeBuilder.setDoorBetweenRooms(rooms.get(30),rooms.get(31));
        mazeBuilder.setDoorBetweenRooms(rooms.get(31),rooms.get(32));
        mazeBuilder.setDoorBetweenRooms(rooms.get(32),rooms.get(42));
        mazeBuilder.setDoorBetweenRooms(rooms.get(42),rooms.get(52));
        mazeBuilder.setDoorBetweenRooms(rooms.get(66),rooms.get(67));
        mazeBuilder.setDoorBetweenRooms(rooms.get(67),rooms.get(68));
        mazeBuilder.setDoorBetweenRooms(rooms.get(68),rooms.get(58));
        mazeBuilder.setDoorBetweenRooms(rooms.get(58),rooms.get(48));
        mazeBuilder.setDoorBetweenRooms(rooms.get(48),rooms.get(38));
        mazeBuilder.setDoorBetweenRooms(rooms.get(38),rooms.get(28));
        mazeBuilder.setDoorBetweenRooms(rooms.get(28),rooms.get(27));
        mazeBuilder.setDoorBetweenRooms(rooms.get(27),rooms.get(26));
        mazeBuilder.setDoorBetweenRooms(rooms.get(26),rooms.get(25));
        mazeBuilder.setDoorBetweenRooms(rooms.get(25),rooms.get(24));
        mazeBuilder.setDoorBetweenRooms(rooms.get(24),rooms.get(23));
        mazeBuilder.setDoorBetweenRooms(rooms.get(66),rooms.get(67));
        mazeBuilder.setDoorBetweenRooms(rooms.get(69),rooms.get(59));
        mazeBuilder.setDoorBetweenRooms(rooms.get(59),rooms.get(49));
        mazeBuilder.setDoorBetweenRooms(rooms.get(49),rooms.get(39));
        mazeBuilder.setDoorBetweenRooms(rooms.get(39),rooms.get(29));
        mazeBuilder.setDoorBetweenRooms(rooms.get(29),rooms.get(19));
        mazeBuilder.setDoorBetweenRooms(rooms.get(19),rooms.get(18));
        mazeBuilder.setDoorBetweenRooms(rooms.get(18),rooms.get(17));
        mazeBuilder.setDoorBetweenRooms(rooms.get(17),rooms.get(16));
        mazeBuilder.setDoorBetweenRooms(rooms.get(16),rooms.get(15));
        mazeBuilder.setDoorBetweenRooms(rooms.get(15),rooms.get(14));
        mazeBuilder.setDoorBetweenRooms(rooms.get(14),rooms.get(4));
        mazeBuilder.setDoorBetweenRooms(rooms.get(14),rooms.get(13));
        mazeBuilder.setDoorBetweenRooms(rooms.get(13),rooms.get(12));
        mazeBuilder.setDoorBetweenRooms(rooms.get(12),rooms.get(22));
        mazeBuilder.setDoorBetweenRooms(rooms.get(22),rooms.get(21));
        mazeBuilder.setDoorBetweenRooms(rooms.get(21),rooms.get(11));
        mazeBuilder.setDoorBetweenRooms(rooms.get(52),rooms.get(53));
        mazeBuilder.setDoorBetweenRooms(rooms.get(53),rooms.get(54));
        mazeBuilder.setDoorBetweenRooms(rooms.get(54),rooms.get(55));
        mazeBuilder.setDoorBetweenRooms(rooms.get(55),rooms.get(56));
        mazeBuilder.setDoorBetweenRooms(rooms.get(56),rooms.get(57));
        mazeBuilder.setDoorBetweenRooms(rooms.get(57),rooms.get(47));
        mazeBuilder.setDoorBetweenRooms(rooms.get(47),rooms.get(37));
        mazeBuilder.setDoorBetweenRooms(rooms.get(37),rooms.get(36));
        mazeBuilder.setDoorBetweenRooms(rooms.get(36),rooms.get(35));
        mazeBuilder.setDoorBetweenRooms(rooms.get(35),rooms.get(34));
```

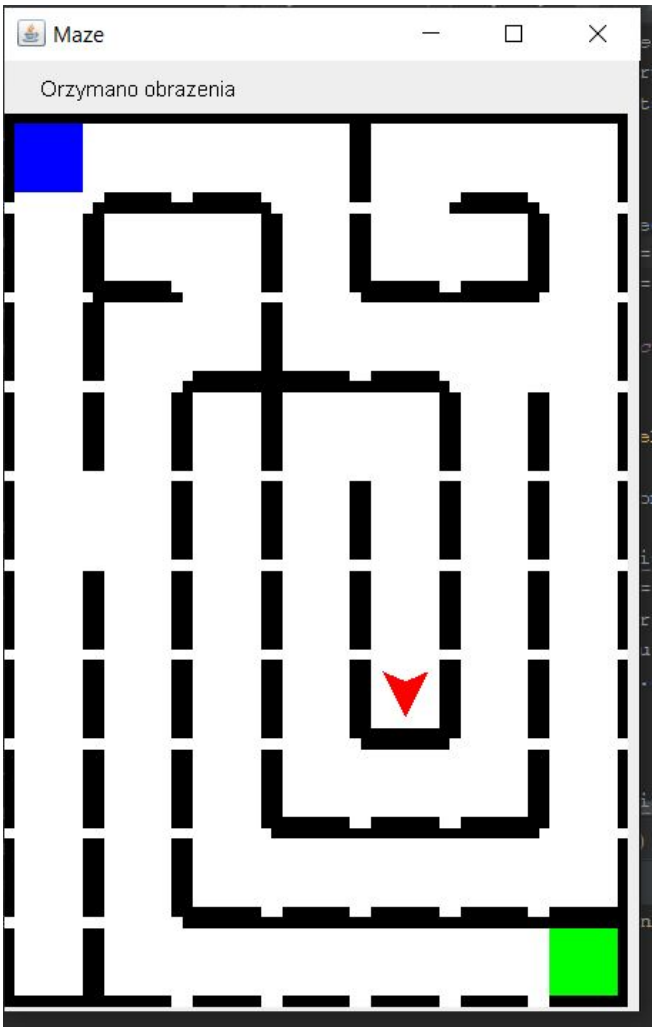
```
        mazeBuilder.setDoorBetweenRooms(rooms.get(34),rooms.get(33));
        mazeBuilder.setDoorBetweenRooms(rooms.get(33),rooms.get(43));
        mazeBuilder.setDoorBetweenRooms(rooms.get(43),rooms.get(44));
        mazeBuilder.setDoorBetweenRooms(rooms.get(44),rooms.get(45));
        mazeBuilder.setDoorBetweenRooms(rooms.get(45),rooms.get(46));

    }
```

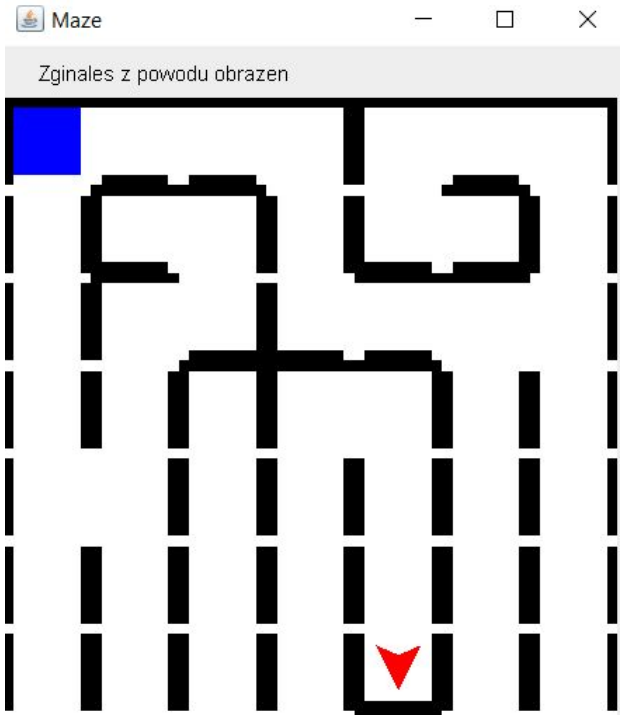
Resultat:






Maze
Uderzyles w magiczna sciane, zaczynasz krwawić


Maze
Orzymano obrazenia


Maze
Zginales z powodu obrazen