Projekt 3

mStudent

Spis treści

1 Treść projektu

Dawno, dawno temu żył sobie student. Większość czasu siedział przed komputerem i grał. Za każdym razem, gry tracił swoje wirtualne życie kopał swój biedny komputer. Pewnego razu podczas gry maszyna zbuntowała się i przeniosła studenta w jego największy koszmar.

... Biedny student musi teraz biegać po rozmaitych labiryntach i zbierać fragmenty kodu, który pozwoli mu na wydostanie się z łap wrednego komputera. Każda plansza zawiera tylko niewielki fragment kodu, który jest zamknięty w sejfie. Ten zaś znajduje się gdzieś w labiryncie przytłuczony tonami kamieni. Nadmiar złego klucz ma jeden z nauczycieli, którzy mają jeden cel – złapać studenta i przykuć go do ławki. Szczęśliwie nasz bohater nie jest pozbawiony „obrony" dysponuje bombami, które rekompilują kod części kamieni i nauczycieli pozostawiając nietknięte klucze i różnego typu dodatkowe umiejętności.

GRA:
Należy utworzyć grę, która przedstawi dalsze losy studenta.

Elementy:

- bomba rekompilująca – rozchodzi się w wszystkich kierunkach na dwa pola, chyba, że napotka na przeszkodę. Czas detonacji jest stały pozwalający na oddalenie się studentowi na pierwszym poziomie gry. Na pole z bombą nie może wejść ani student, ani nauczyciel.

- Klucz – otwiera sejf, jeżeli zostanie zniszczony gracz traci jedno życie i rozpoczyna grę od początku.
- Sejf – jest wyjściem z planszy. Student może wejść na wyższy poziom, jeżeli wejdzie na pole sejfu z kluczem.
- bonus – pojawia się losowo jeden raz na planszę z szansą ¼. Nagroda występuje w dwóch odmianach: przyspieszająca studenta i zwiększająca zasięg bomby o 1 pole. Jeżeli nie zostanie podjęty przez studenta przez X sekund znika.
- ściany – twarde ceglane mury opierające się bombie rekompilującej
- kamienie elementy rozmieszczone na planszy przez twórcę gry. Występują w trzech rodzajach: wapienie, bazalty i granity. Różnią się odpornością na bomby rekompilujące
- nauczyciele – rozpoczynają polowanie na studenta w miejscach koncentracji ustalonych przez twórcę gry. Każdy z nauczycieli może zostać zdekompilowany, w takiej sytuacji może on zostawić klucz lub bonus. Nauczyciel widzi studenta lub bombę, jeżeli znajdują się ona nie dalej niż dwa pola przed nim lub na sąsiadującym polu. Jeżeli student jest oddalony o 4 pola wówczas nauczyciel widzi jedynie kierunek, którym może poszukiwać studenta. Nauczyciel złapie studenta, gdy obaj znajdą się na tym samym polu.
- Student – porusza się w jednym z czterech kierunków wskazanych przez gracza, zostawia bomby i ucieka przed nauczycielami.

Plansza:
prostokątny obszar wypełniony ścianami zgodnie z poniższym schematem, na którym znajdują się elementy gry.

```
xxxxsxxxx
x       x
x x x x x x
x       x
xxxxxxxxx
```

Poziom gry:
Gdy gracz przechodzi na następny poziom zwiększa się rozmiar planszy, a nauczyciele widzą o jedno pole naprzód.

1.1 Elementy dodatkowe

a) Bomby – student dysponuje plecakiem który jest bardzo pojemny wiec postanowiłem dać mu możliwość postawienia więcej niż jednej bomby – nie musi czekać aż postawiona bomba wybuchnie i wtedy dopiero może stawiać następne, od razu może postawić nawet do 14 bomb.
   Każda bomba „żyje" - jej stan się zmienia wraz z upływem czasu, jaki dzieli ja do dekompilacji.
b) Kamienie – skoro różnią się odpornością na bomby, to użytkownik musi widzieć, w jakim stanie po wybuchu są poszczególne kamienie na planszy. Dlatego po każdym wybuchu w kamieniach pojawiają się dziury oraz inne zniszczenia. Mimo że już możemy przejść przez pole, na którym był kamień, widzimy jeszcze pozostałości po kamiennej ścianie. Kolejna bomba postawiona w tym miejscu wyczyści już wszystko i żadnych pozostałości nie będzie.
c) Student – został wyposażony w nowoczesne krzesło odrzutowe pozwalające mu szybkie przemieszczanie się po planszy. Podczas ruchu widzimy jak „ogień z potężnych silników o mocy 200 Kucyków pozostawia smugi na drodze"….
   Krzesło pomaga poruszanie się studentowi po planszy: jeżeli student zamierza np. na najbliższym skrzyżowaniu skręcić w lewo a jeszcze nie „doleciał" do zakrętu to możemy już wcześniej nacisnąć klawisz 4 lub ← i student sam skręci. Pozwala to na sprawniejsze poruszanie się po planszy.
d) Plansza – zrezygnowałem z proponowanego ustawienia ścian:

```
xxxxsxxxx
x       x
x x x x x
x       x
xxxxxxxxx
```

zastąpiłem je takim schematem :

```
x x x x x s x  x  x
x   x   x  x x  x
x              x
x   x  x  x  x  x
x   x  x  x  x  x
x   x  x  x  x  x
x              x
x x   x   x  x  x
x              x
x x x x x x x xx x
```

gdzie x –ściana s – sejf

Przestrzeń, która zostaje wolna po wymurowaniu ścian zostaje wypełniona losowo, losowymi typami kamieni, z warunkiem ze w każdym rzędzie powinno być nie więcej niż 3 kamienie. Ograniczenie dodałem po to żeby nie było zbyt dużo „zabawy z bombkami",
a i tak jest dość do zdetonowania na planszy.

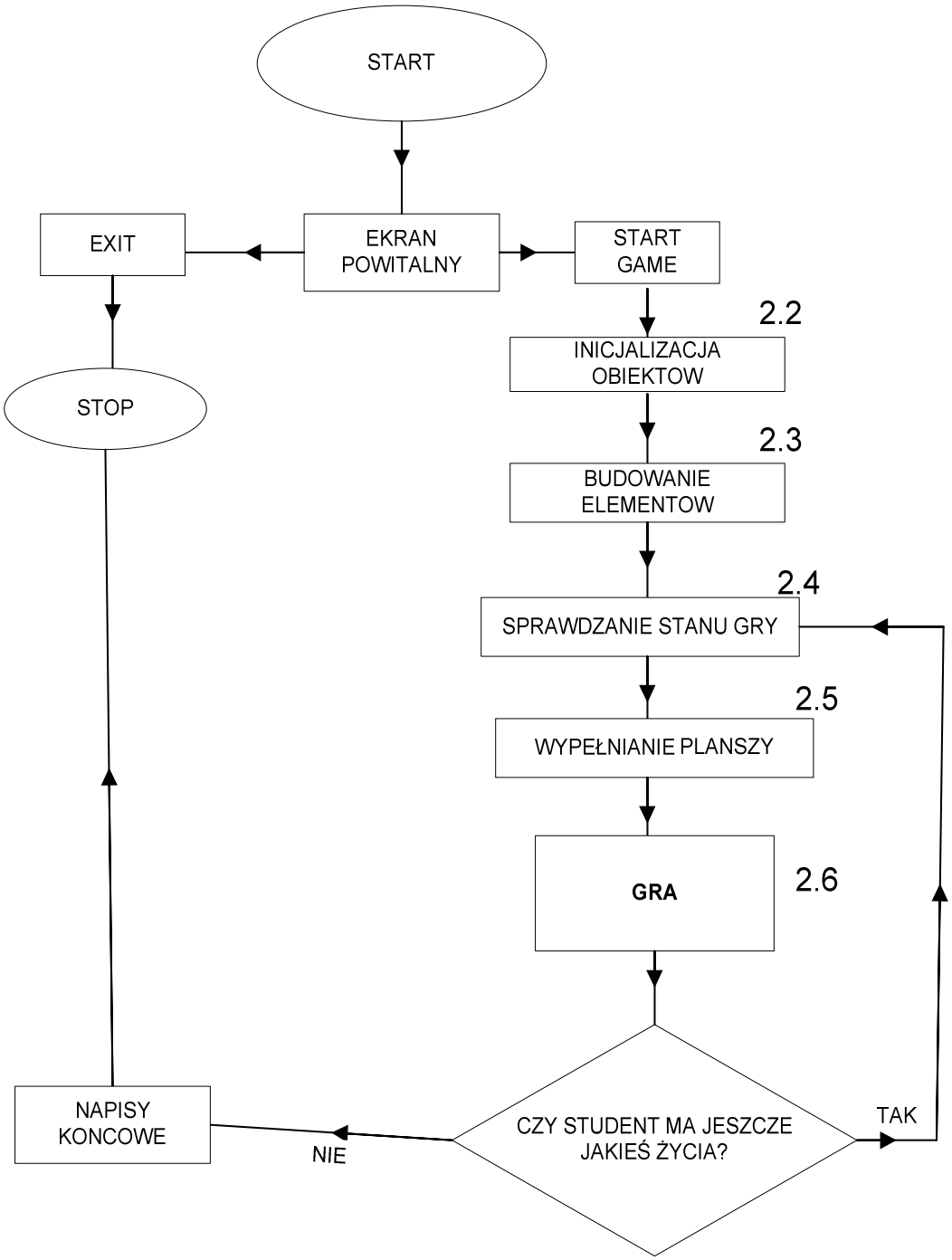
e) Poziomy gry – przewidziałem 4 i chyba nikt nie miałby ochoty grać więcej. Gra kończy się (… ekranem pożegnalnym), gdy osiągniemy 5 poziom lub, gdy stracimy wszystkie z pięciu żyć, jakimi dysponujemy od początku gry.

1.2 Świat studenta – rozwiązanie problemu

Świat studenta składa się z pola gry - odpowiada za to klasa Playground dziedzicząca po Canvas-, na którym „budujemy" sobie poszczególne elementy (ściany, kamienie, student, sejf etc.) Za każdy typ elementu odpowiada osobna klasa, która dziedziczy po klasie Properties. Takie rozwiązanie pozwala mi jako pole gry utworzyć tablice dwuwymiarowa obiektów klasy Properties i łatwe zarządzanie nimi podczas rozgrywki.

2.Algorytm działania

2.1 Diagram główny



2.2 Inicjalizacja obiektów

Tworzy pola klasy Playground tworzy obiekty innych klas np. student (klas Hero), klucz (klasa Key), sejf (klasa Safe) etc.

2.3 Budowanie elementów

Na tym etapie program wczytuje wszystkie potrzebne pliki .png do pamięci.
Tworzone są obiekty Image, które umieszczane są w odpowiednich tablicach. Z tych tablic program będzie pobierał potrzebne mu pliki graficzne.
Jeśli nie znajdzie jakiegoś pliku wyświetli nam informacje na konsoli, o jakiego pliku potrzebował

2.4 Sprawdzanie stanu gry

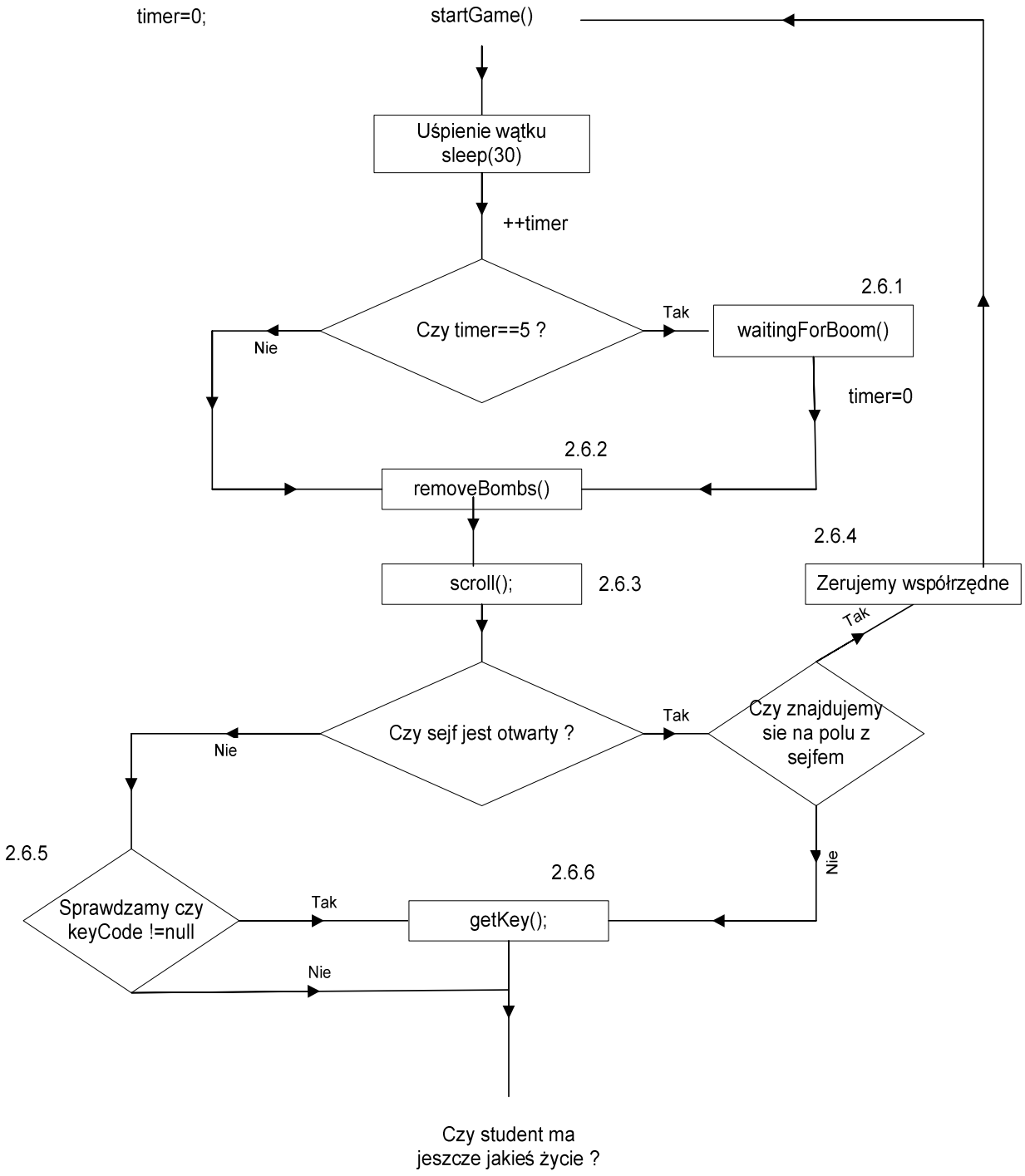
Program sprawdza, na jakim poziomie gry jesteśmy i w zależności od tego dostosowuje rozmiary planszy. Tworzy nowe tablice na postawione bomby (te, które czekają na wybuch).

2.5 Wypełnianie planszy

Program rysuje mur dookoła planszy oraz pojedyncze „filary” w środku, których nie da się zdekompilować. Następnie umieszczany jest sejf (zawsze w tym samym miejscu nie zależnie od poziomu gry). Taki stan planszy wyrysowuje (poprzez metodę createImage()) jako background tworząc obiekt Image. To jest tło, które nie będzie się zmieniac w trakcie gry.

Następnie ustawiamy Kamienie, nauczyciela i przygotowujemy miejsce na naszego bohatera (stawiamy go w takim miejscu żeby mógł się ruszyć i postawić gdzieś bombę nie dekompilujac siebie przy okazji :-) Tam, gdzie ustawiamy jeden z w/w obiektów, automatycznie ustawia się wartość TRUE w tablicy (mapie) refresh[]. W tej tablicy znajdują się tylko wartości true i false, które pokazują nam czy mamy na planszy rysować zawartość danej komórki planszy czy nie. Unikamy wiec rysowania za każdym razem wszystkich obiektów nawet tych, których stan się nie zmienił przed ostatnio wykonana akcja.

2.6 GRA


timer=0;
startGame()
Uśpienie wątku
sleep(30)
++timer
Czy timer==5 ?
Tak
Nie
2.6.1
waitingForBoom()
timer=0
2.6.2
removeBombs()
scroll();
2.6.3
2.6.4
Zerujemy współrzędne
Tak
Czy sejf jest otwarty ?
Tak
Nie
Czy znajdujemy
sie na polu z
sejfem
Nie
2.6.5
Sprawdzamy czy
keyCode !=null
Tak
2.6.6
getKey();
Nie
Czy student ma
jeszcze jakieś życie ?

2.6.1 waitingForBombs();

Metoda „przebiega” po tablicy bomb i włącza odliczanie ustawionych bomba. Podczas odliczania zmieniają się obrazki, które reprezentują bombę. Jeżeli odliczanie dojdzie do zera to bomba wybucha i poprzez metodę makeFire(…) rekompiluje wszystkie elementy, które znajda się w jej zasięgu (oprócz ścian i sejfu). Po wybuchu bomba ustawia znacznik old na true;

2.6.2 removeBombs();

Metoda „przebiega” po tablicy bomb i szuka takich bomb, które maja, old=true jeśli takie SA to znaczy ze już wybuchły i trzeba je usunąć bo już się do niczego nie przydadzą.

2.6.3 scroll();

Metoda sprawdza czy po naciśnięciu klawisza na klawiaturze telefonu student powinien się ruszyć. Jeśli tak to sprawdza kierunek, w którym porusza się student i dostosowuje do kierunku odpowiednią animację studenta.

Jeżeli plansza gry nie mieści się na ekranie to, jeśli student po przekroczeniu środka naszego wyświetlacza, program przesuwa nam plansze w odpowiednim kierunku.

Metoda ta odpowiada również za stawianie bomby przez studenta, kiedy naciśniemy FIRE lub klawisz 5. W tym wypadku tworzy nowy obiekt bombę i dodaje go do tablicy bomb na pierwsze wolne miejsce w tej tablicy (szukając od miejsca 0).

2.6.4 Zerowanie współrzędnych

Następuje, gdy jesteśmy na polu z sejfem- margines błędu 2 pixele. Wtedy zwiększamy poziom gry ustawiamy początkowe współrzędne dla studenta i wywołujemy metodę startGame(), która dostosuje plansze do aktualnego poziomu i wymaluje wszystkie obiekty.

2.6.7 Sprawdzamy czy keyCode !=null

Jeśli keyCode != null oznacza to że na planszy pojawił się klucz (a co za tym idzie jeden z nauczycieli zginął… :-).

Klucz może oczywiście zostać zdekompilowany bombą przez nieuważnego studenta

2.6.6 getKey();

Metoda sprawdza czy przypadkiem nie znaleźliśmy się na polu na którym znajduje się klucz. Jeśli taka sytuacja nastapiła to następuje zmiana właściwości sejfu- na pole z sejfem można wejść a jego atrybut open zmienia się na TRUE;

3.Uwagi

3.1 Problemy z uruchomieniem

Program pisałem w mikroJavie z podłączonym MIDP 2.0 (oczywiście pisałem zgodnie ze specyfikacją 1.0) głownie dlatego że emulator telefonu jest inny tzn. wyświetlany telefon posiada większy wyświetlacz a podczas pisania programu, ciągłych poprawek np. ruchu studenta, większy wyświetlacz był bardzie wygodny.
Natomiast po napisaniu już projektu podpiąłem MIDP 1.0 program się kompiluje ale podczas uruchamiania wyskakuje bład illegalStateException. Metodą prób i błędów doszedłem do wniosku, że emulatorowi nie pasuje linijka sAlert.setCommandListener(this); w klasie student uruchamiającej program dlatego porobiłem tam komentarze co trzeba „odkomentowac" i „zakomentowac" żeby było dobrze tzn. gra się uruchamia ale nie będzie ekranów powitalnych i końcowych.
Jakoś nie mogłem dojść, dlaczego ten błąd się pokazuje tym bardziej, że jak mam „podpięte" MIDP 2.0 lub program odpalam z WTK 2.0 nic podobnego się nie pokazuje.

3.2 Nauczyciel

Niestety nie udało mi się „obsłużyć" nauczyciela tzn. nie dałem mu żadnej inteligencji i w tej wersji jest „statyczny" ale można go zdekompilować i zabrać klucz ☺
Oczywiście próbowałem pisać algorytm, który sprawdza, kiedy nauczyciel ma podążać za „odgłosami" studentem a kiedy gonić go, ale po ponad 300 liniach kodu i milionów if'ów i else'ów i case'ów stwierdziłem że to co pisze to straszna łopatologia a liczba warunków stale rośnie. Po prostu nie potrafiłem jakoś sensownie tego napisać nie miałem pomysłu.

3.3 Obrazki a pamięć telefonu

Obrazków jest dużo, w tym dwa trochę „cięższe" na ekran powitalny i końcowy.
Animacja studenta jest zrobiona za pomocą dwóch klatek (obrazków), bo uznałem, że jak dołożę jeszcze dwie klatki na każdy kierunek poruszania to o ile na naszych superszybkich domowych PC-tach wszystko wygląda ładnie, to już na komórkę może się to nie zmieścić no i jak sądzę czas na załadowanie gry wydłużyłby się znacznie. To samo dotyczy animacji bomb

Mstudent

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class Mstudent extends MIDlet implements CommandListener{
    public Display display;
    public Command exit = new Command("EXIT", Command.EXIT,0);
    public Command newGame = new Command("Start Game", Command.BACK, 1);
    public static Alert sAlert = new Alert("Welcome stranger....");
    public static Alert eAlert = new Alert("Thx 4 game! Next
please...");
    private Ticker title = new Ticker("::mStudent:: by M. Enjoy the
game!");
    private Ticker cast = new Ticker("cast : Student, Bombs, Walls,
Stones and others...Czytal Łucjan Szołański");
    public Image img;
    public Image img2;
    public Mstudent(){
         display = Display.getDisplay(this);
         setStart(); //w razie illegalStateException usunac
         setEnd(); //w razie illegalStateException usunac
    }
    public void startApp(){
        display.setCurrent(sAlert); //w razie illegalStateException
usunac
        //display.setCurrent(new Playground(display)); //w razie
illegalStateException odkomentowac
    }
    public void setStart(){
        try {
             img2 = Image.createImage("/intro.png");
         }catch (Exception exc) {}
         sAlert.setTicker(title);
         sAlert.setImage(img2);
         sAlert.setTimeout(Alert.FOREVER);
         sAlert.addCommand(exit);
         sAlert.addCommand(newGame);
         sAlert.setCommandListener(this);
    }
    public void setEnd(){
        try {
             img = Image.createImage("/outro.png");
         }catch (Exception exc) {}
         eAlert.setTicker(cast);
         eAlert.setImage(img);
         eAlert.setTimeout(Alert.FOREVER);
         eAlert.addCommand(exit);
         eAlert.setCommandListener(this);
    }
    public void pauseApp(){}
    public void destroyApp(boolean unconditional){}

    public void commandAction(Command c, Displayable s) {
     if(c==exit){
```

```
                destroyApp(true);
                notifyDestroyed();
        }
        if(c==newGame) {
             display.setCurrent(new Playground(display));
        }
    }
}
```

Playground

```
import javax.microedition.lcdui.*;
import java.util.*;
import java.lang.*;
import javax.microedition.midlet.*;
public class Playground extends Canvas implements Runnable{

    public int level;
    public static int key = 0;
    public static Properties[][] ground;
    public static Hero student;
    public Image background;
    private int scrWidth = getWidth();
    private int scrHeight = getHeight();
    private static int width;
    private static int height;
    public int visibleX = 0; // wspolrzedne widoczne na LCD
    public int visibleY = 0; // --------------------------
    public static final int SIZE = 15;
    private static int i=0, j=0;
    private int counter = 0;
    public Random rand = new Random();
    public int randx=0;
    public int stonesNo=3;
    public static Playground pg;
    public static int bPixx, bPixy;
    public static boolean[][] refresh;
    public static boolean[] placeForBomb;
    public static Bomb[] bombTable;
    public int timer =0;
    public static boolean deadStudent = false;
    public static boolean gameOver = false;
    public Display d;
    public Safe safe;
    public static Key keyCode;
    public static Teacher teach;
    private boolean white=false;
    public Playground(Display dis){
        d =dis;
        pg = this;
        level = 1; // 1-poczatek gry
        gameBuilder();
        student = new Hero(SIZE, SIZE,5);
        startGame();
        (new Thread(this)).start();

    }
    public void paint(Graphics g){
         if(!white){
            g.setColor(0xffffff);
            g.fillRect(0, 0, scrWidth, scrHeight);
            white=true;
         }
        g.drawImage(background,visibleX, visibleY,
Graphics.TOP|Graphics.LEFT);
```

```
        for(i=0;i < width;++i){
            for(j=0;j < height;++j){
                if(refresh[i][j]) ground[i][j].draw(g, visibleX,
visibleY);
            }
        }
        student.draw(g,visibleX, visibleY);
    }
    protected void keyPressed(int kc) {
        key = kc;
        student.changeStep();
    }
    protected void keyReleased(int kc) {
        key = 0;
        student.changeStep();
    }
    public void run(){
         while(true){
            try{
                Thread.sleep(30);
            }
            catch(Exception e){}
            ++timer;
            if(timer==5){
                waitingForBoom();
                timer=0;
            }
            removeBombs();
            scroll();
            if(safe.open){
                if(student.y <= safe.y+2){
                    ++level;
                    student.x=SIZE;
                    student.y=SIZE;
                    visibleX=0;
                    visibleY=0;
                    startGame();
                }
            }
            else if(keyCode!=null) getKey();
            if(deadStudent){
                repaint();
                try{
                    Thread.sleep(1000);
                }catch(Exception e){}
                startGame();
                student.stateNo=1;
            }
            repaint();
        }
    }
    public void gameBuilder(){ //buduje murki kamienie ticzerow i herosa
        Brick.build();
        Stone.build();
        Safe.build();
        Hero.build();
```

```
        Bomb.build();
        Key.build();
        Teacher.build();
    }
    public void setProperties(){
        background = Image.createImage(width*SIZE, height*SIZE);
        Graphics g = background.getGraphics();
        g.setColor(0xffffff);
        g.fillRect(0,0, width*SIZE, height*SIZE); //malujemy na bialo
tlo

        setWalls();
        setSafe();
        for(i=0;i < width;++i){
            for(j=0;j < height;++j){
                if (ground[i][j]!=null) ground[i][j].draw(g, visibleX,
visibleY);
            }
        }
        setStones();
        setTeacher();
        prepareForOurHero();
    }
    public static void setKey(int tx, int ty){
        keyCode = new Key(tx, ty); //temp
        ground[tx/SIZE][ty/SIZE]=keyCode;
        teach = null;
        refresh[tx/SIZE][ty/SIZE]=true;
    }
    public void setTeacher(){
        teach = new Teacher((width-2)*SIZE, (height-2)*SIZE); //temp
        ground[width-2][height-2]=teach;
        refresh[width-2][height-2]=true;
    }
    public void startGame(){
     if(gameOver) theEnd();
     else{
        if(level==1){
            width = 9;
            height =11;
        }
        else if(level==2){
            width = 13;
            height =11;
        }
        else if(level==3){
            width = 13;
            height =15;
        }
        else if(level==4){
            width = 17;
            height =19;
        }
        else if(level==5){
            gameOver =true;
            startGame();
```

```
        }
        ground = new Properties[width][height];    //glowna tablica
        refresh = new boolean[width][height];
        for(i=0;i < width;++i){ // zeruje tablice elementow do
odswierzania
            for(j=0;j < height;++j){
                refresh[i][j]=false;
            }
        }
        bombTable = new Bomb[15];
        placeForBomb = new boolean[15];
        deadStudent= false;
        for(i=0; i<10; ++i){
            placeForBomb[i]=false;
        }
        setProperties();    //po ustaleniu rozmiaru i gdzie co jest na
planszy
     }
    }
    public void levelUp(){
        ++level;
    }
    public void setWalls(){
        for(i=0;i<width;++i){
            ground[i][0] = new Brick(i*SIZE, 0);
            ground[i][height-1] = new Brick(i*SIZE, (height-1)*SIZE);
        }
        for(i=0;i<height;++i){
            ground[0][i] = new Brick(0, i*SIZE);
            ground[width-1][i] = new Brick((width-1)*SIZE, i*SIZE);
        }
        for(i=0;i<width;i+=2){
            ground[i][height/2] = new Brick(i*SIZE, (height/2)*SIZE);
        }
        for(i=0;i<height;i+=2){
            ground[width/2][i] = new Brick((width/2)*SIZE, i*SIZE);
        }
        for(i=2;i<width; i+=2){
            for(j=2;j<height; j+=2){
                ground[i][j]=new Brick(i*SIZE, j*SIZE);
            }
        }
    }
    public void setStones(){
         for(i=1;i< height-1; ++i){
                while(counter!=stonesNo){
                    randx = Math.abs(rand.nextInt()%(width-1));
                    if(ground[randx][i]==null){
                        ground[randx][i]= new Stone(randx*SIZE, i*SIZE,
(Math.abs(rand.nextInt()%3)+1));
                        refresh[randx][i]= true; //zeby odswiezal
                        counter++;
                    }
                }
                counter = 0;
        }
```

```
    }
    public void setSafe(){
        safe = new Safe(5*SIZE, 0);
        ground[5][0]=safe;
    }
    public void prepareForOurHero(){
        ground[1][1]=null;
        ground[1][2]=null;
        ground[2][1]=null;
        refresh[1][1]=false;
        refresh[1][2]=false;
        refresh[2][1]=false;
    }
    public void scroll(){
        if(student.canGo(key)){
            switch(student.direction){
            case 2:{
                    if( ((visibleX + width*SIZE) > scrWidth) &&
(student.x + visibleX > scrWidth/2) ){
                        visibleX--;
                    }
                    break;
                    }
            case 3:{
                    if( (visibleX < 0) && (student.x + visibleX <=
scrWidth/2) ){
                        visibleX++;
                    }
                    break;
                    }
            case 4:{
                    if( (visibleY < 0) && (student.y + visibleY <=
scrHeight/2) ){
                        visibleY++;
                    }
                    break;
                    }
            case 5:{
                    if( (visibleY + height*SIZE > scrHeight) &&
(student.y + visibleY > scrHeight/2) ){
                        visibleY--;
                    }
                    break;
                    }
        }
        }
    }
    public static void makeFire(int bx, int by, int range){
        bPixx = bx/SIZE;
        bPixy = by/SIZE;
        for(i=1;i<range;++i){
                if(bPixx+i<width && ground[bPixx+i][bPixy]!=null &&
(ground[bPixx+i][bPixy].trespassing==false ||
ground[bPixx+i][bPixy].type==6)){
                    ground[bPixx+i][bPixy].destroy();
                    refresh[bPixx+i][bPixy]=true;
```

```
                }
                if(bPixx-i >=0 && ground[bPixx-i][bPixy]!=null &&
(ground[bPixx-i][bPixy].trespassing==false || ground[bPixx-
i][bPixy].type==6)){
                    ground[bPixx-i][bPixy].destroy();
                    refresh[bPixx-i][bPixy]=true;
                }
                if(bPixy+i<height && ground[bPixx][bPixy+i]!=null &&
(ground[bPixx][bPixy+i].trespassing==false ||
ground[bPixx][bPixy+i].type==6)){
                    ground[bPixx][bPixy+i].destroy();
                    refresh[bPixx][bPixy+i]=true;
                }
                if(bPixy-i >= 0 && ground[bPixx][bPixy-i]!=null &&
(ground[bPixx][bPixy-i].trespassing==false || ground[bPixx][bPixy-
i].type==6)){
                    ground[bPixx][bPixy-i].destroy();
                    refresh[bPixx][bPixy-i]=true;
                }
        }
        System.out.println(student.x + "--"+ bx);
        System.out.println(student.y + "--"+ by);
        if(student.x == bx){
            if( (Math.abs(student.y - by) < (range)*SIZE) ){
               student.boom();
           }
        }
        else if(student.y == by){
            if( (Math.abs(student.x - bx) < (range)*SIZE) ){
               student.boom();
           }
        }
        else{
           if( Math.abs(student.x - bx) + Math.abs(student.y - by) <
(range)*SIZE-2 ){
                student.boom();
           }
        }
         System.out.println(Math.abs(student.x - bx) +
Math.abs(student.y - by));
    }
    public static int findPlaceForBomb(){
        int place=0;
        for(i = 0; i < placeForBomb.length; ++i){
           if(placeForBomb[i]==false) place=i;
        }
    return place;
    }
    public void waitingForBoom(){
        for(i = 0; i < placeForBomb.length; ++i){
           if(placeForBomb[i]) bombTable[i].on();
        }
    }
    public void removeBombs(){
        for(i = 0; i < placeForBomb.length; ++i){
           if(placeForBomb[i] && bombTable[i].old){
```

```
                    int tx = bombTable[i].x/SIZE;
                    int ty = bombTable[i].y/SIZE;
                    System.out.println(tx +" niszczy " +ty);
                    bombTable[i] = null;
                    ground[tx][ty] = null;
                    refresh[tx][ty] = false;
                    placeForBomb[i] = false;
                }
            }
        }
        public void getKey(){
            if(Math.abs(student.x - keyCode.x) + Math.abs(student.y -
keyCode.y) < 2){
                safe.unlock();
                refresh[keyCode.x/SIZE][keyCode.y/SIZE]=false;
                keyCode = null;
            }
        }
        public void theEnd(){
           d.setCurrent(Mstudent.eAlert);
        }
}
```

Tools

```
import javax.microedition.lcdui.*;

abstract class Properties{
    public int power;
    public int type;
    public int x,y;
    public boolean trespassing;
    public boolean ret =false;
    public Properties(int p, int x, int y, boolean t, int whatIsIt){
        power=p;
        trespassing=t;
        this.x=x;
        this.y=y;
        type =whatIsIt;
    }
    abstract public void draw(Graphics g, int x, int y);
    public void destroy(){
        if(trespassing==false && type!=0){
            --power;
            if(power==0){
                trespassing=true;
                if(type == 7){
                    Playground.setKey(x, y);
                }
            }
        }
        else if(trespassing==true && type == 6){
            (Playground.student).boom();
        }
    }
    public void relativeDraw(Graphics g, Image img, int visx, int visy){
        g.drawImage(img, x+visx, y+visy ,Graphics.TOP|Graphics.LEFT);
    }
}
// sciany boczne z cegielek typ-0
class Brick extends Properties{
    public static Image wall;
    public Brick(int x, int y){
       super(100, x, y, false, 0);
    }
    public static void build(){
        try{
           wall = Image.createImage("/brick.png");
        }
        catch(Exception exc){System.out.println("sciany");}
    }
    public void draw(Graphics g, int visx, int visy){
        g.drawImage(wall, x+visx, y+visy ,Graphics.TOP|Graphics.LEFT);
    }
}
//kamienie i ich typ wapien-1 bazalt-2 granit-3
class Stone extends Properties{
    public static Image[] stype1;
    public static Image[] stype2;
```

```
    public static Image[] stype3;
    public static Image[] current;
    public Stone(int x, int y, int type){
       super(type, x, y, false, type);
    }
    public int checkType(){
    return type;
    }
    public static void build(){
        stype1 = new Image[2];
        stype2 = new Image[3];
        stype3 = new Image[4];
            fill(stype1,1);
            fill(stype2,2);
            fill(stype3,3);
    }
    public static void fill(Image[] tab, int t){
        for(int i=0;i<tab.length;++i){
            try{
                tab[i] = Image.createImage("/stone" +t+ "_" +i+ ".png");
            }
            catch(Exception exc){System.out.println("kamyczki");}
        }
    }
    public void draw(Graphics g, int x, int y){
        switch(super.type){
            case 1:{
                current = stype1;
                break;
                }
            case 2:{
                current = stype2;
                break;
                }
            case 3:{
                current = stype3;
                break;
                }
        }
        super.relativeDraw(g, current[type-power], x, y);
    }
}//taki tam sejf typ 0
class Safe extends Properties{
    public boolean open = false;
    public static Image safe;

    public Safe(int x, int y){
        super(100, x, y, false, 0);
    }
    public static void build(){
        try{
           safe = Image.createImage("/safe.png");
        }
        catch(Exception exc){System.out.println("sejfy");}
    }
    public void draw(Graphics g, int visx, int visy){
```

```
        g.drawImage(safe, x+visx, y+visy ,Graphics.TOP|Graphics.LEFT);
    }
    public void unlock(){
        open =true;
        super.trespassing = true;
    }
}//bombki choinkowe
class Bomb extends Properties{
    public int range; //zasieg razenia hehehe
    public int timer; //
    public int step = 0;
    public boolean old =false;
    public static Image[] bombs;
    public Bomb(int x, int y, int ran, int sec){
        super(1, x, y, false, 1);
        range =ran;
        timer = sec;
    }
    public static void build(){
        bombs = new Image[3];
        try{
           bombs[0] = Image.createImage("/bomb_0.png");
           bombs[1] = Image.createImage("/bomb_1.png");
           bombs[2] = Image.createImage("/flame.png");
        }
        catch(Exception exc){System.out.println("bombki choinkowe");}
    }
    public void draw(Graphics g, int visx, int visy){
        super.relativeDraw(g, bombs[step], visx, visy);
    }
    public void on(){
        timer--;
        if(timer%3==0) step=0;
        else step = 1;
        if(timer==0){
           step=2;
           Playground.makeFire(x, y, range);
           old=true;
        }
    }
}//kluczyki i wytrychy :-) typ 6
class Key extends Properties{
    public static Image key;

    public Key(int x, int y){
        super(1, x, y, true, 6);
    }
    public static void build(){
        try{
           key = Image.createImage("/key.png");
        }
        catch(Exception exc){System.out.println("kluczyki");}
    }
    public void draw(Graphics g, int visx, int visy){
        super.relativeDraw(g, key, visx, visy);
    }
```

```
}//ticzer type =7
class Teacher extends Properties{
    public static Image teechair;
    public Teacher(int x, int y){
        super(1, x, y, false, 7);
    }
    public static void build(){
        try{
           teechair = Image.createImage("/teechair.png");
        }
        catch(Exception exc){System.out.println("bad guy");}
    }
    public void draw(Graphics g, int visx, int visy){
        super.relativeDraw(g, teechair, visx, visy);
    }
}
```

Hero

```
import javax.microedition.lcdui.*;
/*
co robi:                    stany:
0 martwy                    2
1 zyje i stoi w miejscu     2
2 prawo                     2
3 lewo                      2
4 gora                      2
5 dol                       2
*/
public class Hero extends Properties{

    public int size;
    public int live;
    public static int stateNo = 1;  //
    public int step = 0; //
    private static int allStates = 6;
    private static int steps = 2;
    public static Image[][] states;
    private boolean ret = false;
    public static int pixx, pixy;
    public static int direction;
    public int timeForBomb = 18; //3 * 6 albo 3* 7 dluzej troche
    public int bombRange = 2; // zwieksza sie potem przy bonusie
    public int bPixx, bPixy;
    public Hero(int x, int y, int liveNo){
        super(100, x, y, true, 0);
        this.x =x;
        this.y =y;
        live = liveNo;
    }
    public static void build(){
        states = new Image[allStates][steps];
        try{
            for(int i = 0; i < allStates; ++i){
                for(int j = 0; j < steps; ++j){
                    states[i][j] = Image.createImage("/hero_" +i+ "_"
+j+ ".png");
                }
            }
        }
        catch(Exception exc){System.out.println("heros");}
    }
    public void changeStep(){
        if(step == 1) step = 0;
        else ++step;
    }
    public boolean canGo(int k){
        ret=false;
        pixx = x/Playground.SIZE;
        pixy = y/Playground.SIZE;
        direction = 0;
        if(k!=0){
```

```
            if(k == Playground.KEY_NUM6 ||
(Playground.pg).getGameAction(k)==Playground.RIGHT){
                if((y % Playground.SIZE)==0){
                    if((x >pixx * Playground.SIZE) ||
((Playground.ground[pixx+1][pixy] ==
null)||(Playground.ground[pixx+1][pixy].trespassing==true))){
                        ret = true;
                        stateNo=2;
                        x++;
                        direction = 2;
                    }
                }
                else{
                    if((Playground.ground[pixx+1][pixy+1] ==
null)||(Playground.ground[pixx+1][pixy+1].trespassing==true)){
                        ret = true;
                        stateNo=5;
                        y++;
                        direction = 5;
                    }
                    else if((Playground.ground[pixx+1][pixy] ==
null)||(Playground.ground[pixx+1][pixy].trespassing==true)){
                        ret = true;
                        stateNo=4;
                        y--;
                        direction = 4;
                    }
                }
            }
            else if(k == Playground.KEY_NUM4 ||
(Playground.pg).getGameAction(k)==Playground.LEFT){
                if((y % Playground.SIZE)==0){
                    if((x >pixx * Playground.SIZE) ||
((Playground.ground[pixx-1][pixy] == null)||(Playground.ground[pixx-
1][pixy].trespassing==true))){
                        ret = true;
                        stateNo=3;
                        x--;
                        direction = 3;
                    }
                }
                else{
                    if((Playground.ground[pixx-1][pixy+1] ==
null)||(Playground.ground[pixx-1][pixy+1].trespassing==true)){
                        ret = true;
                        stateNo=5;
                        y++;
                        direction = 5;
                    }
                    else if((Playground.ground[pixx-1][pixy] ==
null)||(Playground.ground[pixx-1][pixy].trespassing==true)){
                        ret = true;
                        stateNo=4;
                        y--;
                        direction = 4;
                    }
```

```
                }
            }
            else if(k == Canvas.KEY_NUM2 ||
(Playground.pg).getGameAction(k)==Playground.UP){
                if((x % Playground.SIZE)==0){
                    if((y >pixy * Playground.SIZE) ||
((Playground.ground[pixx][pixy-1] ==
null)||(Playground.ground[pixx][pixy-1].trespassing==true))){
                        ret = true;
                        stateNo=4;
                        y--;
                        direction = 4;
                    }
                }
                else{
                    if((Playground.ground[pixx+1][pixy-1] ==
null)||(Playground.ground[pixx+1][pixy-1].trespassing==true)){
                        ret = true;
                        stateNo=2;
                        x++;
                        direction = 2;
                    }
                    else if((Playground.ground[pixx][pixy-1] ==
null)||(Playground.ground[pixx][pixy-1].trespassing==true)){
                        ret = true;
                        stateNo=3;
                        x--;
                        direction = 3;
                    }
                }
            }
            else if(k == Playground.KEY_NUM8 ||
(Playground.pg).getGameAction(k)==Playground.DOWN){
                if((x % Playground.SIZE)==0){
                    if((y >pixy * Playground.SIZE) ||
((Playground.ground[pixx][pixy+1] ==
null)||(Playground.ground[pixx][pixy+1].trespassing==true))){
                        ret = true;
                        stateNo=5;
                        y++;
                        direction = 5;
                    }
                }
                else{
                    if((Playground.ground[pixx+1][pixy+1] ==
null)||(Playground.ground[pixx+1][pixy+1].trespassing==true)){
                        ret = true;
                        stateNo=2;
                        x++;
                        direction = 2;
                    }
                    else if((Playground.ground[pixx][pixy+1] ==
null)||(Playground.ground[pixx][pixy+1].trespassing==true)){
                        ret = true;
                        stateNo=3;
                        x--;
```

```
                        direction = 3;
                    }
                }
            }
            else if(k == Playground.KEY_NUM5 ||
(Playground.pg).getGameAction(k)==Playground.FIRE){ //powies bombke na
choince
                 bPixx = (x + 6)/ Playground.SIZE;
                 bPixy = (y + 6)/ Playground.SIZE;
                 if(Playground.ground[bPixx][bPixy]==null ||
(Playground.ground[bPixx][bPixy].trespassing == true)){
                    Bomb b = new Bomb(bPixx*Playground.SIZE,
bPixy*Playground.SIZE, bombRange, timeForBomb);
                    int p = Playground.findPlaceForBomb();
                    Playground.bombTable[p] = b;
                    Playground.ground[bPixx][bPixy] =
Playground.bombTable[p];
                    Playground.placeForBomb[p] = true;
                    Playground.refresh[bPixx][bPixy] = true;
                    Playground.key=0;
                 }

            }
            else{
                stateNo=1;
                direction = 0;
            }
        }
    return ret;
    }
    public void draw(Graphics g, int visx, int visy){
        super.relativeDraw(g, states[stateNo][step], visx, visy);
    }
    public void boom(){
        live--;
        stateNo=0;
        Playground.deadStudent =true;
        if(live==0) Playground.gameOver =true;

    }
}
```