

Projekt Programowanie Obiektowe w Javie

Grupa:

- Daniel Ostrowski
- Sebastian Zawierucha

Temat: Gra Tower Defense

Biblioteki zewnętrzne znajdują się w folderze “lib”.

Sposób działania

Projekt składa się z 7 package'ów (i w sumie 19 klas) :

- Gra (package dotyczący aplikacji)
 - a) App - Klasa rozpoczynająca grę
 - b) GameLoop - Klasa odpowiadająca za pętlę w grze
 - c) GameMainFrame - Klasa odpowiadająca za główne okno
 - d) GamePanel - Klasa w której jest wykonywany cały gameplay
 - e) GameState - Klasa informująca o stanie aplikacji u przeciwnika
- Images (package dotyczący grafiki)
 - a) Image - Typ wyliczeniowy, w którym są napisane nazwy obrazków
 - b) ImageFactory - Klasa, która dopisuje URL obrazka do nazwy z “Image”
- Math (package dotyczący operacji matematycznych w grze)
 - a) Calculation - Klasa która zajmuje się obliczeniami takim jak np. kolizja
- Model (package dotyczący modeli wieżyczek czy przeciwników)
 - a) Sprite - Klasa abstrakcyjna przechowująca dane i metody związane z przeciwnikami
 - b) Tower - Klasa abstrakcyjna przechowująca dane i metody związane z wieżyczkami
 - c) Balloon, FastBalloon, SlowBalloon - klasy z przeciwnikami
 - d) FastTowerTierI, ObszarTower, SlowTower - klasy z wieżyczkami
- Siec (pakiet dotyczący komunikacji między graczami)
 - a) Communication - Klasa odpowiada za wysyłanie i odbieranie danych między graczami
- Sounds (pakiet dotyczący dźwięków)
 - a) PlayMusic - Klasa korzystająca z zewnętrznej biblioteki “jaco-mp3-player-0.9.3” do odtwarzania dźwięku (u nas jest to soundtrack)
- Stale (pakiet zawiera stałe użyte w programie)
 - a) Constants - klasa zawierająca stałe statyczne

1. Połączenie

Po odpaleniu gry wyskoczy nam okno, gdzie należy podać ip drugiego gracza, jeśli drugi gracz znajduje się na tej samej maszynie należy zostawić sugerowane ip. "127.0.0.1". Po zatwierdzeniu wyskoczy drugie okno pytające nas czy jesteś hostem, odpowiadamy wpisując "true" lub "false" i zatwierdzając. Ważne jest aby host odpalił grę jako pierwszy a w drugiej kolejności klient. Gra wykorzystuje dwa porty 7777 oraz 7778.

2. Sterowanie

Sterowanie w tej grze opiera się na klawiaturze i myszce.

- Mysz (wybierasz miejsce, gdzie chcesz postawić wieżyczkę lub zdecydować, którą wieżyczkę usunąć)
- Klawiatura
 - a) "1" - Tryb kupujący (Należy go wybrać jeśli chcemy stawiać wieżyczki lub kupować dodatkowych przeciwników drugiemu graczowi)
 - b) "2" - Tryb sprzedający (Należy go wybrać jeśli chcemy sprzedać jakąś wieżyczkę)
 - c) "q" - Wybór szybkiej wieżyczki
 - d) "w" - Wybór wolnej mocnej wieżyczki
 - e) "e" - Wybór obszarowej wieżyczki
 - f) "a" - Wybór kupna przeciwnika normalnego dla drugiego gracza
 - g) "s" - Wybór kupna przeciwnika wytrzymałego, wolnego dla drugiego gracza
 - h) "d" - Wybór kupna przeciwnika szybkiego dla drugiego gracza

3. Rozgrywka

Po połączeniu, rozpoczyna się rozgrywka. Każdy z graczy ma do dyspozycji 100 golda na start. Pieniądze w grze można zdobywać poprzez niszczenie przeciwników, natomiast wydawać je można na wieżyczki lub dodatkowych przeciwników dla drugiego gracza. Należy mądrze wydawać zarobione złoto, gdyż o porażkę nie ciężko. Wieżyczki można postawić kiedy (1. jesteśmy w trybie kupującym, 2. gdy wieżyczka znajduje się na obszarze dozwolonym do budowy)

4. Koniec gry

Kiedy jeden z graczy przegra, to serwer poinformuje o tym drugiego gracza.



Najważniejsze fragmenty kodu

1. Mechanizm przesyłania informacji.

```
92      UPnP.openPortUDP(7777);|
93
94      if(Objects.equals( a: "true", czyHost))
95          isHost = true;
96      else if(Objects.equals( a: "false", czyHost))
97          isHost = false;
98      try {
99          connection = new Communication( portNumber: 7777,ip, isHost);
100     } catch (IOException e) {
101         e.printStackTrace();
102     }
```

92 - odblokowanie portu UDP 7777

94 - 97 - na podstawie odpowiedzi użytkownika ustawia zmienną isHost na true or false

99 - utwórz obiekt klasy Communication jako argumenty przyjmuje port, ip drugiego gracza oraz zmienną typu bool wskazującą czy gracz jest hostem, klasa odpowiada za komunikację między graczami

```
47      public GameState receiveGameState() throws IOException, ClassNotFoundException {
48
49          byte[] buffer = new byte[65535];
50
51          DatagramPacket datagramPacket = new DatagramPacket(buffer, buffer.length);
52          socket.setSoTimeout(2000);
53          socket.receive(datagramPacket);
54          ObjectInputStream inputStream = new ObjectInputStream(new ByteArrayInputStream(buffer));
55          GameState gs = (GameState) inputStream.readObject();
56
57          return gs;
58      }
59
60
61      public void sendGameState(GameState gs) throws IOException{
62
63          ByteArrayOutputStream out = new ByteArrayOutputStream();
64          ObjectOutputStream outputStream = new ObjectOutputStream(out);
65          outputStream.writeObject(gs);
66          outputStream.close();
67
68          byte[] buffer = out.toByteArray();
69
70          DatagramPacket datagramPacket = new DatagramPacket(buffer, buffer.length, InetAddress.getByName(ipAdress), portNumber);
71
72          socket.send(datagramPacket);
73      }
```

47 - metoda klasy Communication odpowiedzialna za odbieranie informacji od drugiego gracza

51-53 tworzony jest obiekt klasy DatagramPacket, następnie ustawiany jest maksymalny czas oczekiwania na 2s, oraz rozpoczęcie oczekiwania na wiadomość.

54-57 - odebrany pakiet jest tłumaczony i zwracany w postaci obiektu

61 - metoda odpowiedzialna jest za wysyłanie obiektu klasy GameState do drugiego gracza

63-72 - podany jako argument obiekt konwertowany jest do postaci tablicy bajtów a następnie wysyłany

```
272         if(isHost)
273         {
274
275             try {
276                 enemyGameState = connection.receiveGameState();
277                 connection.sendGameState(gameState);
278
279             } catch (IOException | ClassNotFoundException e) {
280                 e.printStackTrace();
281             }
282         }
283         else
284         {
285             try {
286                 connection.sendGameState(gameState);
287                 enemyGameState = connection.receiveGameState();
288             } catch (IOException | ClassNotFoundException e) {
289                 e.printStackTrace();
290             }
291         }
```

272-291 - fragment kodu umieszczony jest w metodzie update(), to czy gracz jest hostem albo klientem wpływa na kolejność wykonywania metod obiektu connection. Host najpierw oczekuje na informacje od klienta po czym wysyła swój stan gry do niego, natomiast klient najpierw wysyła informacje po czym czeka na odpowiedź ze strony hosta. Rozwiązanie to pozwala na łatwe ominięcie problemu desynchronizacji, ale powoduje, że gdy jeden gracz ma słabe łącze gra zwalnia dla obu graczy.

```

11 public class GameState implements Serializable {
12
13     private int playerHp;
14     private int buyBalloon;
15     private List<Sprite> balloonList;
16     private List<Tower> towerList;
17
18     public GameState(int playerHp, List<Sprite> balloonList, List<Tower> towerList, int buyBalloon){
19
20         this.playerHp = playerHp;
21         this.balloonList = balloonList;
22         this.towerList = towerList;
23         this.buyBalloon = buyBalloon;
24     }
25
26     public void addOffset(){
27
28         for(Sprite balloon : this.balloonList) {
29             balloon.setX(balloon.getX() + 650);
30         }
31
32         for(Tower tower : this.towerList) {
33             tower.setX(tower.getX() + 650);
34         }
35     }
36
37     public int getPlayerHp() { return this.playerHp; }
38
39     public List<Sprite> getBalloonList() { return balloonList; }
40
41     public List<Tower> getTowerList() { return towerList; }
42
43     public int getBuyBalloon() { return buyBalloon; }
44
45 }

```

Klasa GameState - w niej przechowywane są informacje jakie chcemy wysłać do drugiego gracza, posiada też metodę która przesuwa położenie wież/przeciwników aby możliwe było wyświetlenie ich u gracza po drugiej stronie planszy.

playerHP - stan naszego zdrowia

buyBalloon - informacja o wysłanych przez nas dodatkowych falach do przeciwnika

balloonList - lista z balonami na naszym ekranie

towerList - lista posiadanych wież

2. Gameplay

```
private void initialize() {  
    addKeyListener(this);  
    addMouseListener(this);  
    addMouseMotionListener(this);  
  
    setFocusable(true);  
    setPreferredSize(new Dimension(Constants.WINDOW_WIDTH, Constants.WINDOW_HEIGHT));  
}
```

Inicjalizacja listenerów dla myszki, klawiatury i akcji.

```
138     protected void paintComponent(Graphics g)
```

paintComponent(Graphics g) - Metoda służąca do wyświetlania obrazów graficznych (w niej są wszystkie grafiki, które mają się wyświetlać).

```
198     public void doneLoop() {  
199  
200         if(player1Hp > 0 && youWin==0) {  
201             update();  
202             repaint();  
203         }  
204         else {  
205             if(youWin==0) {  
206                 showLose = 1;  
207                 repaint();  
208             }  
209             else {  
210                 repaint();  
211             }  
212         }  
213     }  
214  
215 }  
216 }
```

(198-216) - tu wykonywana jest pętla, która wywołuje metodę update (opisana poniżej) oraz repaint służącą do ponownego załadowania grafiki (we właściwych miejscach).

```

218 private void update() {
219     if(PlayMusic.musicIsStopped())
220         PlayMusic.playMusic();
221
222     if(ticsToNewWave==1000) {
223         ticsToNewWave=0;
224         wave++;
225     }
226
227     if(ticsToNewWave==0) {
228         for(int i=0; i<5*wave;i++) {
229             SlowBalloon slowballoon = new SlowBalloon(50,-50-(50*i));
230             this.player1balloonList.add(slowballoon);
231
232             Balloon balloon = new Balloon(50,-50-(50*i));
233             this.player1balloonList.add(balloon);
234
235             FastBalloon fastBalloon = new FastBalloon(50,-50-(50*i));
236             this.player1balloonList.add(fastBalloon);
237         }
238     }
239
240     ticsToNewWave++;
241 }

```

(219-220) - jeżeli soundtrack się skończył zaczyna odtwarzać go od nowa

(222-225) - jeżeli ticsToNewWave (ilość wykonanych update'ów) będzie równa 1000 to zwiększa wave (aktualna fala) o 1.

(227-238) - jeżeli ticsToNewWave==0 to generowana jest nowa fala z większą ilością przeciwników.

```

242     for(Sprite balloon : this.player1balloonList) {
243         if(balloon.isVisible()) {
244             player1Gold+=balloon.move();
245             if(balloon.getY() == 600)
246                 player1Hp=player1Hp-2;
247         }
248     }
249
250     for(Tower tower : this.player1towerList) {
251         tower.shoot(player1balloonList);
252     }
253
254     for(Sprite balloon : this.player1balloonList) {
255         if(balloon.getY() >= 700 || !balloon.isVisible()) {
256             this.player1balloonList.remove(balloon);
257             break;
258         }
259     }
260 }

```

(242-248) - jeżeli przeciwnik żyje to wykonuje metoda move(), a jeżeli przeciwnik doszedł do y==600 (czyli przeszedł przez całą ścieżkę) to zabiera graczowi HP.

(250-252) - tu wykonywana jest operacja strzelania wieżyczek. (w klasie tower sprawdzane jest czy przeciwnik znajduje się w zasięgu wieżyczki.

(254-259) - jeżeli y przeciwnika będzie większe bądź równe 700 i nie żyje to zostanie usunięty z listy przeciwników.

```
313         if(ticsToNewWave>1) {  
314             if(enemyGameState.getPlayerHp()<=0)  
315                 youWin=1;  
316         }  
317     }
```

(313-316) - jeżeli zdrowie przeciwnika spadnie poniżej 0 to zmienia youWin na 1 co powoduje wyświetlenie obrazka "YOU WIN".

```
325 @Override  
326 public void keyPressed(KeyEvent e) {  
327     if(e.getKeyCode() == KeyEvent.VK_1) {  
328         isBuying = true;  
329         isSelling = false;  
330     }  
331     if(e.getKeyCode() == KeyEvent.VK_2) {  
332         isBuying = false;  
333         isSelling = true;  
334     }  
335     if(e.getKeyCode() == KeyEvent.VK_Q) {  
336         typeTower = 1;  
337     }  
338     if(e.getKeyCode() == KeyEvent.VK_W) {  
339         typeTower = 2;  
340     }  
341     if(e.getKeyCode() == KeyEvent.VK_E) {  
342         typeTower = 3;  
343     }
```

(327-334) - ustawianie tryby kupującego lub sprzedającego.

(335-343) - wybór wieżyczki do postawienia

```

344         if(e.getKeyCode() == KeyEvent.VK_A) {
345             if(player1Gold >= 50 && buyedBaloons == 0) {
346                 buyedBaloons = 1;
347                 player1Gold = player1Gold - 50;
348             }
349         }
350         if(e.getKeyCode() == KeyEvent.VK_S) {
351             if(player1Gold >= 200 && buyedBaloons == 0) {
352                 buyedBaloons = 3;
353                 player1Gold = player1Gold - 200;
354             }
355         }
356         if(e.getKeyCode() == KeyEvent.VK_D) {
357             if(player1Gold >= 100 && buyedBaloons == 0) {
358                 buyedBaloons = 2;
359                 player1Gold = player1Gold - 100;
360             }
361         }
362     }

```

(344-361) - wybór przeciwnika do kupienia dla drugiego gracza.

```

376     @Override
377     public void mouseMoved(MouseEvent e) {
378         xMouse = e.getX();
379         yMouse = e.getY();
380         isCursorON= Calculation.collisonWithRoad(e.getX(), e.getY(), 50, 50, 0) && this.player1towerList.isEmpty();
381         for(Tower tower : this.player1towerList) {
382             if(Calculation.collison(Calculation.distance(e.getX(), e.getY(), 0, 0, tower.getX(), tower.getY(), 50, 50)) || Calcul
383                 isCursorON=true;
384                 break;
385             }
386             else {
387                 isCursorON=false;
388             }
389         }
390     }

```

(378-379) - zapisywanie aktualnego położenia myszki w aplikacji

(380-389) - Sprawdzanie czy dochodzi do kolizji wieżyczki z innym obiektem (ogólnie metoda sprawdza czy można postawić w danym miejscu wieżyczkę.

```

404 @Override
405 public void mousePressed(MouseEvent e) {
406     if(isBuying==true && e.getX() <= 600) {
407         if(isCursorON==false && player1Gold >=50) {
408             if(typeTower==1) {
409                 player1towerList.add(new FastTowerTierI(e.getX()-25,e.getY()-25));
410                 player1Gold=player1Gold-50;
411                 isCursorON=true;
412             }
413             else if(typeTower==2 && player1Gold >=100) {
414                 player1towerList.add(new SlowTower(e.getX()-25,e.getY()-25));
415                 player1Gold=player1Gold-100;
416                 isCursorON=true;
417             }
418             else if(typeTower==3 && player1Gold >=150) {
419                 player1towerList.add(new ObszarTower(e.getX()-25,e.getY()-25));
420                 player1Gold=player1Gold-150;
421                 isCursorON=true;
422             }
423         }
424     }
425 }

```

(405-425) - metoda stawiająca wieżyczkę w miejscu kursora o ile nie zachodzi tam kolizja, która na to nie pozwala.

```

426     if(isSelling==true && e.getX() <= 600 && isCursorON==true) {
427         for(Tower tower : this.player1towerList) {
428             if(Calculation.distance(e.getX(), e.getY(), 0, 0, tower.getX(), tower.getY(), 50, 50) < 50 && isCursorON==true) {
429                 player1towerList.remove(tower);
430                 break;
431             }
432         }
433         isCursorON=true;
434     }
435 }

```

(426-435) - metoda usuwająca wieżyczkę, na której znajduje się kursor.

```

295     switch (enemyGameState.getBuyBalloon()){
296
297         case 1:
298             for(int i = 0; i < 5; i++){
299                 Balloon balloon = new Balloon( x: 50, y: -50-(50*i));
300                 this.player1balloonList.add(balloon);
301             }
302             break;
303         case 2:
304             for(int i = 0; i < 5; i++){
305                 FastBalloon fastBalloon = new FastBalloon( x: 50, y: -50-(50*i));
306                 this.player1balloonList.add(fastBalloon);
307             }
308             break;
309         case 3:
310             for(int i = 0; i < 5; i++){
311                 SlowBalloon slowballoon = new SlowBalloon( x: 50, y: -50-(50*i));
312                 this.player1balloonList.add(slowballoon);
313             }
314             break;
315         default:
316             break;
317     }

```

295-317 - w zależności od wybranej przez przeciwnika opcji, do naszej gry zostaną dodani dodatkowi przeciwnicy

3. Kolizja i metody obliczające.

```

10 public static double distance(int x1, int y1, int width1, int height1, int x2, int y2, int width2, int height2 ){
11     double distance;
12     double centerX1 = x1 + (width1/2.0);
13     double centerY1 = y1 + (height1/2.0);
14     double centerX2 = x2 + (width2/2.0);
15     double centerY2 = y2 + (height2/2.0);
16
17     distance = sqrt(pow(centerX2 - centerX1,2) + pow(centerY2 - centerY1,2));
18
19     return distance;
20 }

```

(10-20) - metoda zwracająca dystans między 2 obiektami

```

22 public static boolean isCursorOver(int cursorX, int cursorY, int objectX, int objectY, int width, int height ){
23
24     if(cursorX >= objectX && cursorX <= objectX + width && cursorY >= objectY && cursorY <= objectY + height){
25         return true;
26     }
27     else {
28         return false;
29     }
30 }

```

(22-30) - metoda sprawdzająca czy kursor znajduje się w obrębie danego obiektu (wewnątrz niego)

```

32 public static boolean collision(double distance){
33     if(distance < 50){
34         return true;
35     }
36     else
37         return false;
38 }

```

(32-38) - metoda sprawdzająca kolizję

```

48 public static boolean collisionWithRoad(int cursorX, int cursorY, int width, int height, int offset ){
49     if((cursorX+(width/2) > 50+offset && cursorX-(width/2) < 100+offset) && cursorY-(height/2) < 200)
50         return true;
51     else if((cursorX+(width/2) > 50+offset && cursorX-(width/2) < 200+offset) && (cursorY-(height/2) < 200 && cursorY+(height/2) < 200))
52         return true;
53     else if((cursorX+(width/2) > 150+offset && cursorX-(width/2) < 200+offset) && (cursorY-(height/2) < 200 && cursorY+(height/2) < 200))
54         return true;
55     else if((cursorX+(width/2) > 150+offset && cursorX-(width/2) < 450+offset) && (cursorY-(height/2) < 100 && cursorY+(height/2) < 100))
56         return true;
57     else if((cursorX+(width/2) > 400+offset && cursorX-(width/2) < 450+offset) && (cursorY-(height/2) < 350 && cursorY+(height/2) < 350))
58         return true;
59     else if((cursorX+(width/2) > 250+offset && cursorX-(width/2) < 450+offset) && (cursorY-(height/2) < 350 && cursorY+(height/2) < 350))
60         return true;
61     else if((cursorX+(width/2) > 250+offset && cursorX-(width/2) < 300+offset) && (cursorY-(height/2) < 500 && cursorY+(height/2) < 500))
62         return true;
63     else if((cursorX+(width/2) > 50+offset && cursorX-(width/2) < 300+offset) && (cursorY-(height/2) < 500 && cursorY+(height/2) < 500))
64         return true;
65     else if((cursorX+(width/2) > 50+offset && cursorX-(width/2) < 100+offset) && (cursorY-(height/2) < 700 && cursorY+(height/2) < 700))
66         return true;
67     else if((cursorY+(height/2) >= 600))
68         return true;
69     else if((cursorX+(width) >= 575))
70         return true;
71     else
72         return false;
73 }
74
75 }

```

(48-75) - metoda sprawdzająca czy doszło do kolizji ze ścieżką.

4. Działanie wieżyczek i przeciwników (balonów)

```
21 public int move() {
22     if (this.y <= 150 && line == 0) {
23         this.y=this.y+dx;
24     }
25     if (this.y == 150 && this.x <= 160) {
26         line = 1;
27         this.x=this.x+dx;
28     }
29     if (this.x == 150 && line == 1) {
30         this.y=this.y-dx;
31     }
32     if (this.y == 50 && this.x <= 400 && line == 1) {
33         this.x=this.x+dx;
34     }
35     if (this.y <= 300 && this.x == 400) {
36         this.y=this.y+dx;
37     }
38     if (this.y == 300 && this.x >= 250) {
39         this.x=this.x-dx;
40         line = 2;
41     }
42     if (this.x == 250 && this.y <= 450 && line == 2) {
43         this.y=this.y+dx;
44     }
45     if (this.y == 450 && this.x >= 50) {
46         this.x=this.x-dx;
47         line = 3;
48     }
49     if (this.x == 50 && this.y < 700 && line == 3)
50         this.y=this.y+dx;
51     if (this.y == 700 && this.x < 600) {
52         line = 0;
53     }
54
55     if(this.hp <= 0){
56         this.setVisible(false);
57         return gold;
58     }
59     return 0;
60 }
```

(21-60) - metoda odpowiadająca za ruch balonów ścieżce

```
68 public void die(){
69     this.dead = true;
70 }
71
72 public void setX(int x){
73     this.x = x;
74 }
75
76 public void setY(int y){
77     this.y = y;
78 }
79
80 public int getX(){
81     return this.x;
82 }
83
84 public int getY(){
85     return this.y;
86 }
87
88 public boolean isDead(){
89     return this.dead;
90 }
91
92 public void setHp(int hp){
93     this.hp = hp;
94 }
95
96 public void dealDMG(int dmg){
97     this.hp -= dmg;
98 }
```

(68-70) - metoda ustawia zmienną odpowiadającą za śmierć dla danego przeciwnika

(72-78) - metody ustawia x i y przeciwnika

(80-86) - metody, które pobierają aktualne x i y przeciwnika

(88-90) - metoda sprawdzająca czy przeciwnik jest martwy

(92-94) - metoda ustawia hp przeciwnika

(96-98) - metoda służąca do zadawania obrażeń przeciwnikowi

```

99
100 public boolean isVisible() {
101     return visible;
102 }
103
104 public void setVisible(boolean visible) {
105     this.visible = visible;
106 }
107
108 public Image getImage(){
109     return this.graphic;
110 }
111
112 }

```

(100-102) - metoda sprawdza czy przeciwnik jest widoczny

(104-106) - metoda ustawiająca widoczność przeciwnika

(108-110) - metoda zwracająca grafikę przeciwnika

```

19 public void shoot(List<Sprite> list) {
20     coolDown--;
21     if(this.coolDown <= 0) {
22         for(int i=0;i<list.size();i++) {
23             if(list.get(i).isVisible()==true && Calculation.distance(this.x, this.y, 50, 50, list.get(i).x, list.get(i).y, 50,
24                 list.get(i).dealDMG(dmg);]
25             coolDown = coolDownAmount;
26             break;
27         }
28     }
29 }
30 }
31

```

(19-30) - metoda wykonująca strzał dla danej wieżyczki do najbliższego przeciwnika w zasięgu.

```
35 public void setX(int x){
36     this.x = x;
37 }
38
39 public void setY(int y){
40     this.y = y;
41 }
42
43 public int getX(){
44     return this.x;
45 }
46
47 public int getY(){
48     return this.y;
49 }
50
51 public Image getImage(){
52     return this.graphic;
53 }
54
55 public int GetRange(){
56     return this.range;
57 }
58
59 }
```

(35-41) - metody ustawiające x i y wieżyczki

(43-49) - metody pobierające aktualne x i y

(51-53) - metoda pobierając grafikę wieżyczki

(55-57) - metoda pobierająca zasięg wieżyczki

```

7   public class ImageFactory {
8
9   @ public static ImageIcon createImage(Image image) {
10      ImageIcon imageIcon = null;
11
12      switch(image) {
13      case MAPA:
14          imageIcon = new ImageIcon(Constants.MAPA_IMAGE_URL);
15          break;
16      case IKONA:
17          imageIcon = new ImageIcon(Constants.IKONA_IMAGE_URL);
18          break;
19      case B1:
20          imageIcon = new ImageIcon(Constants.PRZECIWNIK_IMAGE_URL);
21          break;
22      case B2:
23          imageIcon = new ImageIcon(Constants.SZYKI_PRZECIWNIK_IMAGE_URL);
24          break;

```

```

52      case T3icon:
53          imageIcon = new ImageIcon(Constants.OBASZAROWA_WIEZYCZKAICON_IMAGE_URL);
54          break;
55      case YouLose:
56          imageIcon = new ImageIcon(Constants.YOULOSE_IMAGE_URL);
57          break;
58      case YouWin:
59          imageIcon = new ImageIcon(Constants.YOUWIN_IMAGE_URL);
60          break;
61      case InterfaceBackground:
62          imageIcon = new ImageIcon(Constants.INTERFACEBACKGROUND_IMAGE_URL);
63          break;
64
65      default:
66          break;
67      }
68
69      return imageIcon;
70  }
71  }

```

Klasa posiada jedną statyczną metodę `createImage()`, która w zależności od podanego argumentu zwraca nam obiekt `imageIcon` z odpowiednią grafiką, ścieżki do grafik przechowywane są w klasie `Constants`

Uwagi i wnioski:

Naszym celem było stworzenie projektu, który da możliwość łatwej rozbudowy i implementacji nowych funkcjonalności. Tower defence jest gatunkiem, który można rozbudowywać w nieskończoność o nowe typy wież czy przeciwników, dlatego staraliśmy się jak najbardziej wykorzystać obiektowość języka Java.