Programowanie III

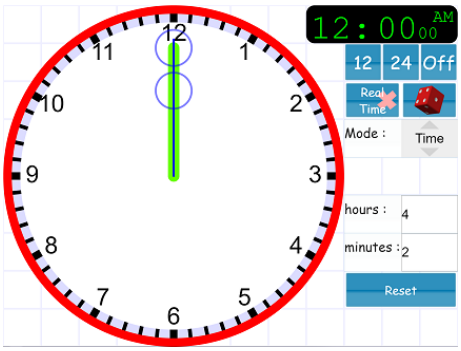
dokumentacja projektu Zegar 2010

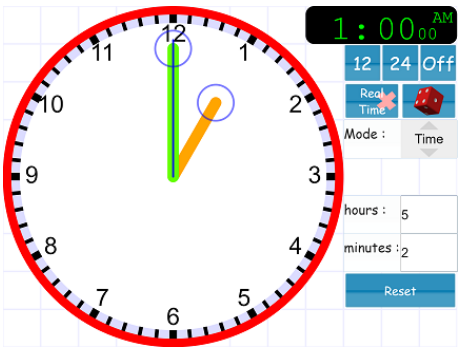
Marceli Pychyński, grupa 3F

Informatyka, wydział Matematyki Stosowanej, sem. III

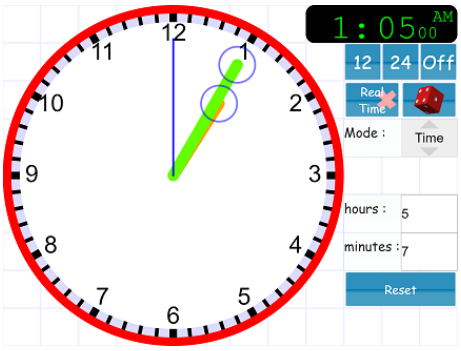
23 grudnia 2020

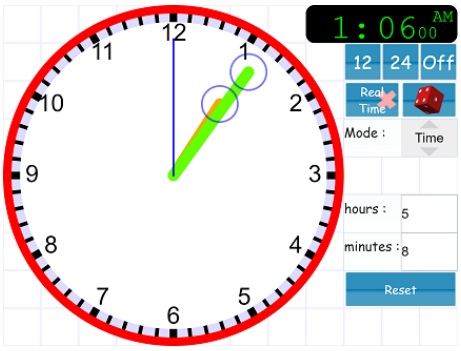
Spis treści





Rysunek 1: Porównanie zegarów w godzinach 12:00 i 13:00





Rysunek 2: Porównanie zegarów w godzinach 13:05 i 13:06

1 Opis problemu przedstawionego w zadaniu

1.1 Treść zadania

W zadaniu rozważamy zegar, który wskazuje na godzinę 12:00. Należy obliczyć ile minie sekund, do ponownego ustawienia się wskazówek w jednej linii. Największym problem jest to, że wskazówki pokonują drogę po okręgu w różnym czasie, a wskazówka minutowa „goni”/podąża za wskazówką godzinową. Dalsza część opisu problemu zawarta jest w modelu matematyczny, ponieważ w trakcie rozważań wyprowadzany jest wzór potrzebny do rozwiązania zadania.

2 Model Matematyczny

2.1 Problem

W ciągu 12h wskazówka godzinowa pokonuje cały okrąg, zatem w 1h pokona 1/12 okręgu, ponieważ na wskazówce minutowej, przejście całego okręgu zajmuje 12h. Z tego wynikają dane (m – minutowa, g – godzinowa): $t_m = 1h$, $t_g = 12h$

W takim razie, po 1h będzie 13:00. Wskazówki będą od siebie w „odległości” 5 minut (godzinowa na: 1, minutowa na: 0), więc po 5 minutach wskazówka minutowa zbliży się bardzo blisko do wskazówki godzinowej, ale nie najdą na

siebie,ponieważ w ciągu tych 5 minut wskazówka godzinowa pokona 2.5 stopnia (wynika to z prędkości kątowej (360°)/12h).

$$s_m \neq s_g$$

Aby wyprowadzić wzór na miejsce spotkania, można do drogi wskazówki godzinowej dodać 2, ponieważ pokonała drogę krótsza o przynajmniej jeden okrąg.

$$s_m = s_g + 2\pi$$

Jednak bez obliczenia, nie jesteśmy w stanie określić dokładnie, na której sekundzie wskazówki ustawią się w jednej linii.

Po krótkiej analizie można uznać, że najlepszą drogą jest tak naprawdę zakreślony kąt (czyli $s_m = \alpha_m$ i $s_g =_g$, a takie dane z jakimi się spotkaliśmy można wykorzystać do zastosowania wzoru na prędkość kątową

Ze wzoru na prędkość kątową $\omega = \frac{\alpha}{t}$, można wyprowadzić wzór $\alpha = \omega * t$

ω – prędkość kątowa
α – zakreślony kąt
t – czas pokonania

Teraz zamieniamy $s_m = s_g + 2\pi$ na:

$$\alpha_m = \alpha_g + 2\pi$$

$\alpha_m = \omega_m t$, gdzie „t" jest szukanym czasem, po którym wskazówki się pokrywają

$$\omega_m t = \omega_g t + 2\pi$$

$\omega = \frac{2\pi}{t}$, a więc:

$$\begin{gathered}\frac{2\pi}{t_m}t = \frac{2\pi}{t_g}t + 2\pi \\ \frac{t}{t_m} = \frac{t}{t_g} + 1 \\ t(\frac{1}{t_m} - \frac{1}{t_g}) = 1 \\ t = \frac{1}{\frac{1}{t_m} - \frac{1}{t_g}}\end{gathered}$$

Teraz do wprowadznego wzoru można podstawić dane, które zostały ustalone na początku rozwiązania:

$$\begin{gathered}t = \frac{1}{\frac{1}{1} - \frac{1}{12}} \\ t = \frac{12}{11}(h)\end{gathered}$$

Gdzie t jest szukanym czasem.

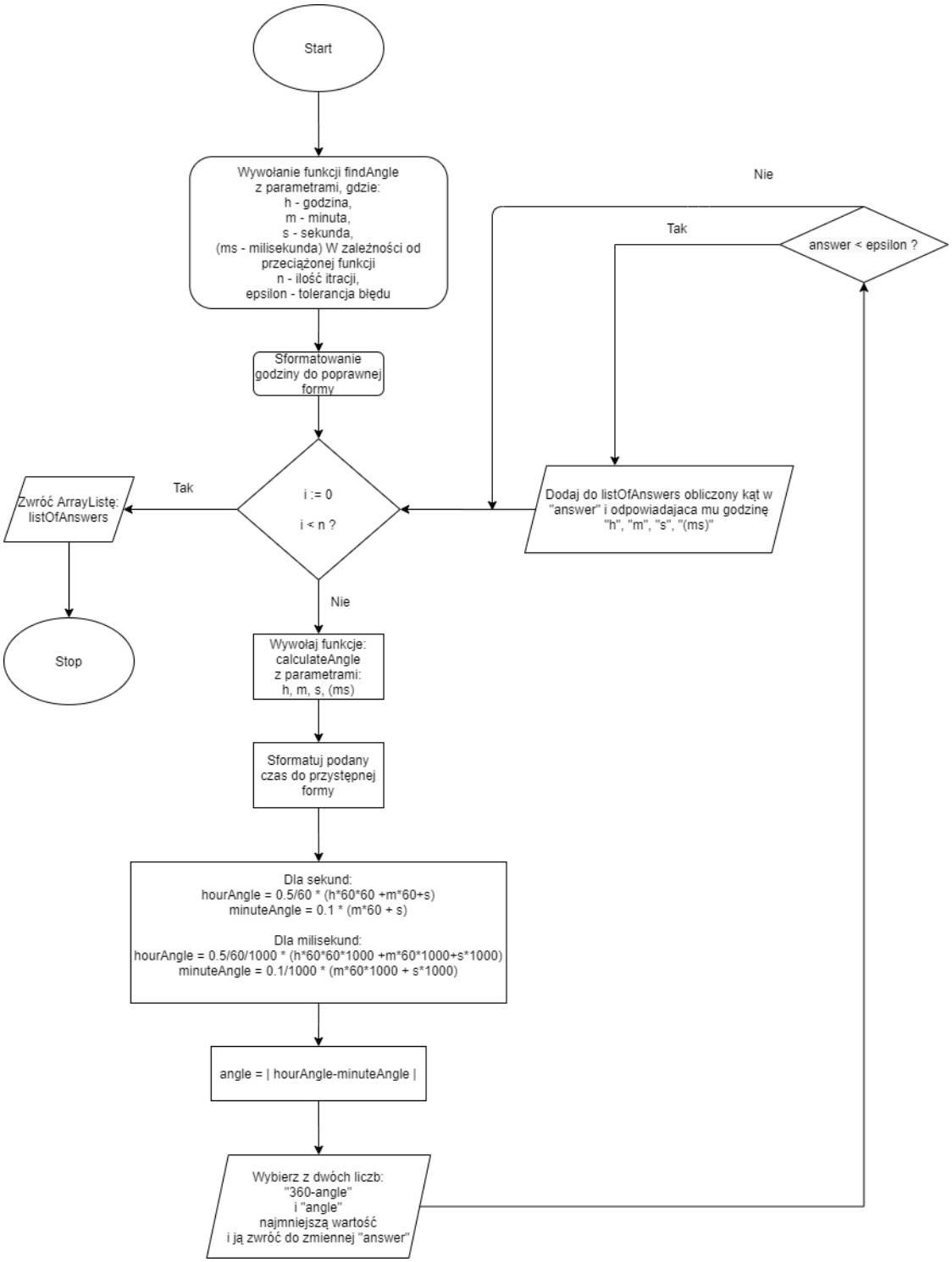
2.2 Problem - metoda numeryczna

Zdecydowałem się na wprowadzenie rozwiązania numerycznego danego problemu, ponieważ po wyprowadzeniu wzoru otrzymujemy dokładne rozwiązanie czyli 12/11 h = 1h 5 m (27) s. Z takim rozwiązanie wykorzystywanie języka programowania nie na sensu.

Moja metoda opiera się na sprawdzaniu co sekundę lub milisekundę jaki kąt został utworzony między wskazówką godzinową, a minutową. Czym kąt jest niższy tym większe prawdopodobieństwo, że wskazówki ustawiły się w jednej linii.

3 Algorytm

Schemat blokowy



4 Implementacja

Wyżej opisany algorytm jest zaimplementowany w postaci 2 funkcji: findAngle i calculateAngle, które są odpowiednio przeciążane w zależności od wymaganej dokładności, przy obliczeniach z dokładnością co do sekundy wystarcza użycie zmiennej double, przy dokładności co do 1 ms wykorzystałem klasę BigDecimal.

Clock.java

```
1  import java.math.BigDecimal;
2  import java.util.ArrayList;
3
4  class Clock {
5
6      //dokladnosc do 1 s
7      static double calculateAngle(double h, double m, double s){
8          if(s == 60) {
9              m += 1;
10             s = 0;
11         }
12         if(m==60){
13             h += 1;
14             m =0;
15         }
16         if(h==12)
17             h=0;
18         if(h>12)
19             h = h-12;
20
21         //katy wskazowek obliczane dla konkretnej godziny minuty i
               sekundy
22         double hourAngle = 0.5/60 * (h*60*60 +m*60+s); //kat wskazowki
               godzinowej
23
24         // 6/60 = 0.1
25         double minuteAngle = 0.1 * (m*60 + s); //kat wskazowki minutowej
26
27         double angle = Math.abs(hourAngle-minuteAngle);
28         angle = Math.abs(Math.min(360-angle,angle));
29         return angle;
30     }
31
32     //dokladnosc do 1 ms
33     static BigDecimal calculateAngle(double h, double m, double s,
           double ms){
34         if(ms == 1000) {
35             s += 1;
36             ms = 0;
37         }
38         if(s == 60) {
39             m += 1;
40             s = 0;
41         }
42         if(m==60){
```

```
43              h += 1;
44              m =0;
45          }
46          if(h==12)
47              h=0;
48          if(h>12)
49              h = h-12;
50
51
52          BigDecimal hAngle = new BigDecimal(0.5/60/1000 * (h*60*60*1000 +
                m*60*1000+s*1000));
53          //double hourAngle = 0.5/60/1000 * (h*60*60*1000 +m*60*1000+s
                *1000); //kat wskazowki godzinowej
54
55          BigDecimal mAngle = new BigDecimal(0.1/1000 * (m*60*1000 + s
                *1000));
56          //double minuteAngle = 0.1/1000 * (m*60*1000 + s*1000); //kat
                wskazowki minutowej
57
58          BigDecimal rAngle;
59
60
61          BigDecimal help = new BigDecimal(360);
62          rAngle = new BigDecimal(String.valueOf((hAngle.subtract(mAngle).
                abs())));
63          rAngle = (rAngle.min(help.subtract(rAngle))).abs();
64
65          //rAngle = Math.abs(Math.min(360-angle,angle));
66          return rAngle;
67      }
68
69      // dokladnosc do 1 s
70      // h,m,s - odpowiada wybranej godzinie, n - ilosc sekund ktore ma
            minac w pomiarze, epsilon - dokladnosc bledu
71      static ArrayList findAngle(int h, int m, int s, int n, double
            epsilon){
72          ArrayList listOfAnswers = new ArrayList<Double>();
73
74          for (int i = 0; i < n; i++){
75              double answer = calculateAngle(h, m, s); //oblicza kat
76              //decyduje czy wybrana godzina spelnia
77              if (answer < epsilon){
78
79                  //zamiana sekund na konkretna godzine
80                  int sec = s;
81                  int min = m;
82                  int hou = h;
83                  while (sec > 60){
84                      sec -=60;
85                      min += 1;
86                  }
87                  while (min > 60){
88                      min -=60;
89                      hou += 1;
90                  }
```

```
91
92
93                  String answerText = ("Kat " + answer + " odpowiada
                        godzinie: " + hou + " h " + min + " m " + sec + " s")
                        ;
94                  listOfAnswers.add(answerText);
95              }
96              s++;
97          }
98          return listOfAnswers;
99      }
100
101     // dokladnosc do 1 ms
102     // h,m,s - odpowiada wybranej godzinie, n - ilosc sekund ktore ma
            minac w pomiarze, epsilon - dokladnosc bledu
103     static ArrayList findAngle(int h, int m, int s, int ms, int n,
            double epsilon){
104         ArrayList listOfAnswers = new ArrayList<Double>();
105
106         for (int i = 0; i < n; i++){
107             BigDecimal answer = calculateAngle(h, m, s, ms); //oblicza
                    kat
108             //decyduje czy wybrana godzina spelnia
109             if (answer.compareTo(BigDecimal.valueOf(epsilon)) == -1){
110
111                 //zamiana sekund na konkretna godzine
112                 int mil = ms;
113                 int sec = s;
114                 int min = m;
115                 int hou = h;
116                 while (mil >1000){
117                     mil -= 1000;
118                     sec += 1;
119                 }
120                 while (sec > 60){
121                     sec -=60;
122                     min += 1;
123                 }
124                 while (min > 60){
125                     min -=60;
126                     hou += 1;
127                 }
128
129
130                 String answerText = ("Kat " + answer + " odpowiada
                        godzinie: " + hou + " h " + min + " m " + sec + " s "
                         + mil + " ms");
131                 listOfAnswers.add(answerText);
132             }
133             ms++;
134         }
135         return listOfAnswers;
136     }
137
138
```

```
139     public static void main(String[] args) {
140
141         // Wypisuje kat (tylko mniejszy niz zadana dokladnosc jako
                epsilon) miedzy wskazowkami zegara  jaki powstaje w ciagu 2h
                (7200 sekund od godz. 12:00)
142
143         int h = 12;
144         int m = 0;
145         int s = 0;
146         int n = 7200;
147
148         ArrayList list, list2, list3;
149         list = findAngle(h,m,s,n,0.1);
150         for (Object x : list){
151             System.out.println(x);
152         }
153
154         System.out.println("\nKolejne wyniki po poprawieniu dokladnosci
                na podstawie wczesniejszych wyliczen\n");
155
156         //wybieramy dokladnosc wieksza, czyli liczbe mniejsza od kat
                dla godziny 12:00:01
157         list2 = findAngle(h,m,s+1,n-1,0.09);
158         for (Object x : list2){
159             System.out.println(x);
160         }
161
162         System.out.println("\nPo okresleniu, ze wskazowki pokrywaja sie
                miedzy godzina 13:05, a godzina 13:06" +
163                 " sprawdzamy katy przy kazdej milisekundzie\n");
164
165         //sprawdzamy kat dla kazdej milisekundy miedzy 13:05:27 a
                13:05:28
166         list3 = findAngle(13,5,27,0, 1000,0.025);
167         for (Object x : list3){
168             System.out.println(x);
169         }
170
171         //Niestety nawet z uzyciem klasy BigDecimal nie jestesmy w
                stanie otrzymac wiekszej dokladnosci
172         // i wyliczyc milisekundy dla ktorej kat bylby wyraznie nizszy
                niz dla godzin 13:05:27 i 13:05:28
173     }
174 }
```

5 Podsumowanie

5.1 Co zostało zrobione

Program pozwala numeryczne obliczyć kąt powstały między wskazówkami. Obliczony kąt musi być jak najmniejszy - jak najbliższy zera.

Kąt 0.0 odpowiada godzinie: 12 h 0 m 0 s
Kąt 0.024999999999977263 odpowiada godzinie: 13 h 5 m 27 s
Kąt 0.024999999999998578914528479799628257751464843 75 odpowiada godzinie: 13 h 5 m 27 s 0 ms

Niestety obliczony kąt dla każdej milisekundy nie zszedł poniżej 0.024999... stopnia i udało się oszacować, że wskazówki pokryją się ponownie około 13:05:27. Dla dokładnych obliczeń jesteśmy w stanie wyliczyć, że będzie to 13:05:27 i 272 milisekundy.

5.2 Pomysły dotyczące rozbudowy

Ciekawym pomysłem jest stworzenie animacji z uzyćiem biblioteki JavaFX, zależną od danej godziny, minuty i sekundy która pokazywałaby jak wskazówki poruszają się po tarczy zegara.