CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

Project ini dapat memvisualisasikan data pada data structure hash table menjadi grafis 2D secara dinamis dengan inputan yang ditentukan oleh user. Data yang di inputkan akan masuk ke data structure hash table, di proses dan mendapatkan nilai yang kemudian dikirim ke user interface. Pada code user interface nilai yang dihasilkan dari data structure digunakan untuk draw dan menentukan koordinat dari visualisasi.

5.1.1 Set Size of Array

```
16      public HashTableMain(int ts) {
17          size = 0;
18          TABLE_SIZE = ts;
19          table = new HashTableList[TABLE_SIZE];
20          for (int i = 0; i < TABLE_SIZE; i++)
21              table[i] = null; // looping table tapi masih kosong
22      }
```

Illustration 5.1: Set Size of Array

(16) The user input will go to int ts, (18) ts for set size of TABLE_SIZE. (19) HashTableList is a list of arrays, each array of TABLE_SIZE will have a list. (21) Number of arrays will be loaded according to TABLE_SIZE with null initial value.

5.1.2 Insert Value

```
39      public void insertvalue(int value)
40      {
41          int hash = (value % TABLE_SIZE);
42          if (table[hash] == null)
43              table[hash] = new HashTableList(value);
44          else
45          {
46              HashTableList entry = table[hash];
47              while (entry.next != null)
48                  entry = entry.next;
49
50              //while (entry.value == value)
51              //entry.value = value;
52
53              if (entry.value == value)
54                  entry.value = value;
55              else
56                  entry.next = new HashTableList(value);
57
58
59          }
60          size++;
61      }
```

Illustration 5.2: Insert Value

(39) The user input will go to int value, (41) the value will modulo with TABLE_SIZE to get location from value. (42) If the initial array is empty then the value will be printed on array list 0. (46) If the initial array is not null then the array will have a list. (47) A loop will be performed to check the location until it gets an empty location and then the data will be printed.

5.1.3 Get Location of Value

```
64      public int[] getLoc(int value) {
65          int hash = (value % TABLE_SIZE);
66          int[] result = {0, 0};//00
67          int[] notFound = {100, 100};
68          int count = 0;
69          if (table[hash] != null) {
70              HashTableList entry = table[hash];
71              result[0] = hash;
72              while (entry != null) {
73                  if (entry.value == value) {
74                      result[1] = count;
75                      //Log.i("index2", String.valueOf(count));
76                      break;
77                  } else
78                      count++;
79                  entry = entry.next;
80              }
81              return result;
82          } else return notFound;
83      }
```

Illustration 5.3: Get Location of Value

(64) Each input value will get the location, the value of the input will go to the int value. (65) value will be modulo with TABLE_SIZE to determine the draw location. (69) If table [modulo result] is not null then table will create list. (71) result [0] is the result to determine the position of x, derived from value modulo with TABLE_SIZE. (72) if next list in each array is not null then count will increase. (74) result [1] comes from the count value. (81) if the process is complete then result is {hash, count}, result is function to get coordinate location x, y at the time of data in draw.

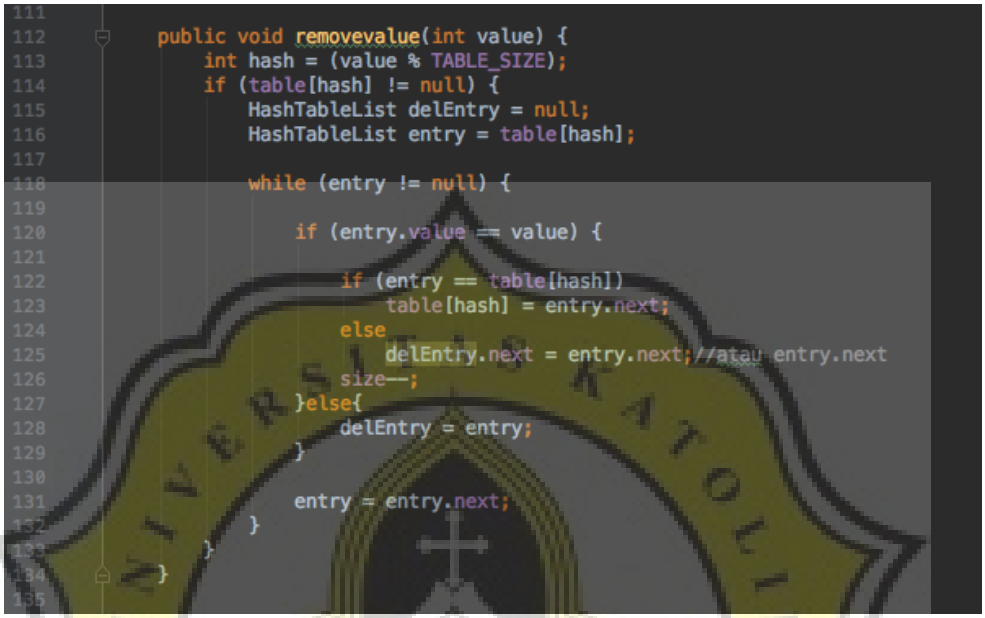
5.1.4 Search and Get Value Location

```
85
86      public int[] getLoc2(int value) {
87          int hash = (value % TABLE_SIZE);
88          int[] result = {1000, 1000};//00
89          int[] notFound = {100, 100};
90          int count = 0;
91          if (table[hash] != null) {
92              HashTableList entry = table[hash];
93
94
95
96              while (entry != null) {
97                  if (entry.value == value) {
98                      result[0] = hash;
99                      result[1] = count;
100                     //Log.i("index2", String.valueOf(count));
101                     break;
102                 } else
103                     count++;
104                 entry = entry.next;
105             }
106             return result;
107         } else return notFound;
108     }
109
```

Illustration 5.4: Search and Get Value Location

(86) Each input value will get the location, the value of the input will go to the int value. (87) value will be dimodulo with TABLE_SIZE to determine the location of draw. (91) If table [module result] is not null then array will have list. (97) if the data from the list is equal to value then result x is the result of modulo and result y is count. (103) If the data from the list is not equal to the value count will increase and (104) will go to the next entry. (106) If data is found it will return to result with result {hash, count} or coordinate x, y. (88) if the data is not found then the result will default to {1000,1000} which means that x will be in the 1000 and y positions at 1000. This value is unlikely to be used in the simulation because the visualized data are limited in coordinates {12.4 }.

5.1.5 Detele Value

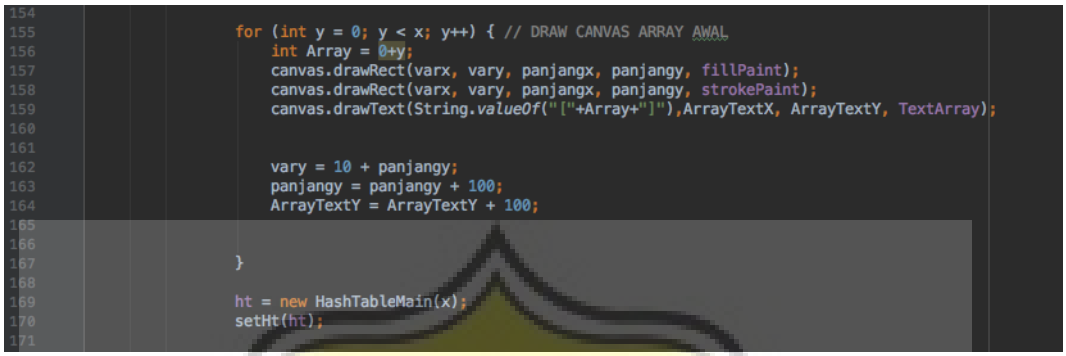


```
111
112     public void removevalue(int value) {
113         int hash = (value % TABLE_SIZE);
114         if (table[hash] != null) {
115             HashTableList delEntry = null;
116             HashTableList entry = table[hash];
117
118             while (entry != null) {
119
120                 if (entry.value == value) {
121
122                     if (entry == table[hash])
123                         table[hash] = entry.next;
124                     else
125                         delEntry.next = entry.next;//atau entry.next
126                     size--;
127                 }else{
128                     delEntry = entry;
129                 }
130
131                 entry = entry.next;
132             }
133         }
134     }
135
```

Illustration 5.5: Delete Value

(112) The value that will be removed by the user will go to int value. (113) value modulo with TABLE_SIZE to determine the coordinates of the value location. (114) if table [result modulo] not null then will go to next process. (120) if the entry value is equal to value delete then check back (122) if the entry is the same as the [modulo] table then (123) the [modulo result] table will be replaced with the value after it. (125) If the entry value is not equal to the detele value then delEntry.next will become null.

5.1.6 Draw Rectangle bassed Array Size

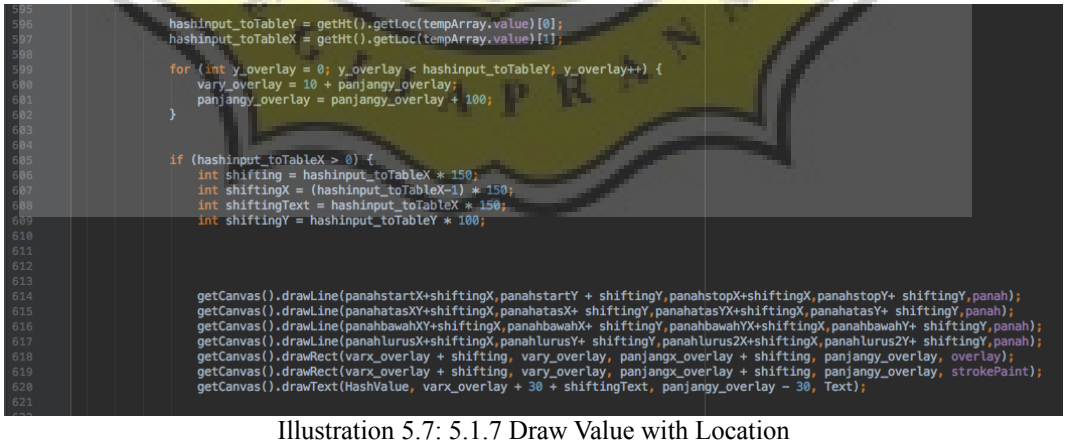


```
154
155                 for (int y = 0; y < x; y++) { // DRAW CANVAS ARRAY AWAL
156                     int Array = 0+y;
157                     canvas.drawRect(varx, vary, panjangx, panjangy, fillPaint);
158                     canvas.drawRect(varx, vary, panjangx, panjangy, strokePaint);
159                     canvas.drawText(String.valueOf("["+Array+"]"),ArrayTextX, ArrayTextY, TextArray);
160
161
162                     vary = 10 + panjangy;
163                     panjangy = panjangy + 100;
164                     ArrayTextY = ArrayTextY + 100;
165
166
167                 }
168
169                 ht = new HashTableMain(x);
170                 setHt(ht);
171
```

Illustration 5.6: 5.1.6 Draw Rectangle bassed Array Size

(112) The value that will be removed by the user will go to int value. (113) value modulo with TABLE_SIZE to determine the coordinates of the value location. (114) if table [result modulo] not null then will go to next process. (120) if the entry value is equal to value delete then check back (122) if the entry is the same as the [modulo] table then (123) the [modulo result] table will be replaced with the value after it. (125) If the entry value is not equal to the detele value then delEntry.next will become null.

5.1.7 Draw Value with Location



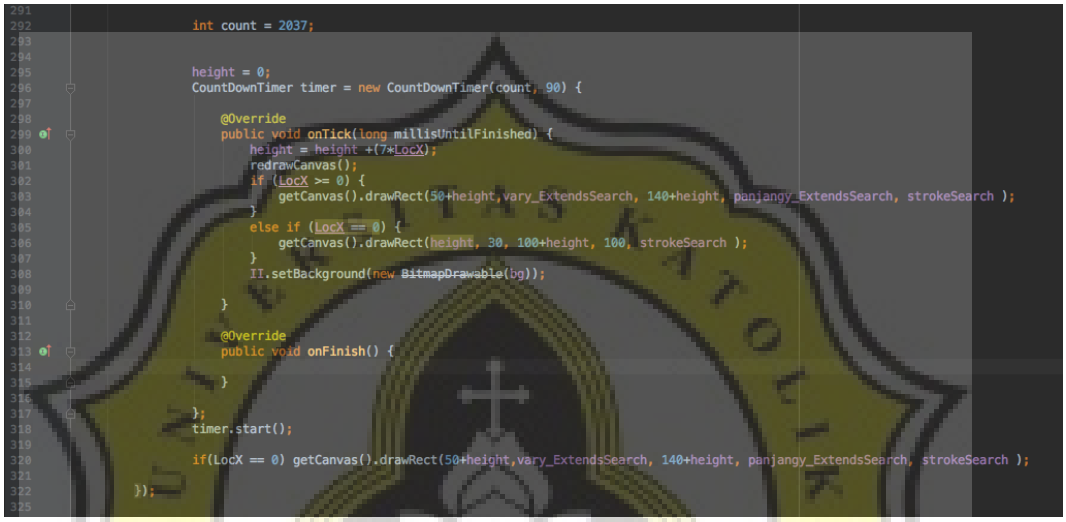
```
595
596             hashinput_toTableY = getHt().getLoc(tempArray.value)[0];
597             hashinput_toTableX = getHt().getLoc(tempArray.value)[1];
598
599             for (int y_overlay = 0; y_overlay < hashinput_toTableY; y_overlay++) {
600                 vary_overlay = 10 + panjangy_overlay;
601                 panjangy_overlay = panjangy_overlay + 100;
602             }
603
604
605             if (hashinput_toTableX > 0) {
606                 int shifting = hashinput_toTableX * 150;
607                 int shiftingX = (hashinput_toTableX-1) * 150;
608                 int shiftingText = hashinput_toTableX * 150;
609                 int shiftingY = hashinput_toTableY * 100;
610
611
612
613
614                 getCanvas().drawLine(panahstartX+shiftingX,panahstartY + shiftingY,panahstopX+shiftingX,panahstopY+ shiftingY,panah);
615                 getCanvas().drawLine(panahatasXY+shiftingX,panahatasX+ shiftingY,panahatasYX+shiftingX,panahatasY+ shiftingY,panah);
616                 getCanvas().drawLine(panahbawahXY+shiftingX,panahbawahX+ shiftingY,panahbawahYX+shiftingX,panahbawahY+ shiftingY,panah);
617                 getCanvas().drawLine(panahlurusX+shiftingX,panahlurusY+ shiftingY,panahlurus2X+shiftingX,panahlurus2Y+ shiftingY,panah);
618                 getCanvas().drawRect(varx_overlay + shifting, vary_overlay, panjangx_overlay + shifting, panjangy_overlay, overlay);
619                 getCanvas().drawRect(varx_overlay + shifting, vary_overlay, panjangx_overlay + shifting, panjangy_overlay, strokePaint);
620                 getCanvas().drawText(HashValue, varx_overlay + 30 + shiftingText, panjangy_overlay - 30, Text);
621
```

Illustration 5.7: 5.1.7 Draw Value with Location

Earlier from getLoc process get result {hash, count} or {x, y}. (596) hashinput_toTableY is the result of the hash or x coordinate of the getLoc process on the hashtable data structure. (597) hashinput_toTableX is the result of

coordinate count or y of the getLoc process in the hashtable data structure. (599 - 609) The above looping function for determining the distance between rectangles with the main parameters is the coordinates x, y.

5.1.8 Animated Rectangle



```
291
292                 int count = 2037;
293
294
295                 height = 0;
296                 CountDownTimer timer = new CountDownTimer(count, 90) {
297
298                     @Override
299                     public void onTick(long millisUntilFinished) {
300                         height = height +(7*LocX);
301                         redrawCanvas();
302                         if (LocX >= 0) {
303                             getCanvas().drawRect(50+height,vary_ExtendsSearch, 140+height, panjangy_ExtendsSearch, strokeSearch );
304                         }
305                         else if (LocX == 0) {
306                             getCanvas().drawRect(height, 30, 100+height, 100, strokeSearch );
307                         }
308                         II.setBackground(new BitmapDrawable(bg));
309
310                     }
311
312                     @Override
313                     public void onFinish() {
314
315                     }
316
317                 };
318                 timer.start();
319
320                 if(LocX == 0) getCanvas().drawRect(50+height,vary_ExtendsSearch, 140+height, panjangy_ExtendsSearch, strokeSearch );
321
322             });
325
```

(292) int count = 2037 is countdown timer. (296) there are 2 variables ie countdown timer and divider. If 2037 and divider 90 then the result is 22.633 or about 2 second. (300) height is a variable to determine the position of the rectangle shift, in about 2 seconds the rectangle will shift as much as 7 pixels multiplied by the x coordinates. For example if the coordinates x = 1 then in 2 seconds the rectangle will shift as much as 7 x 1 (y coordinates) x 22,633 = 158,431. The search animation will shift from pixel 0 to pixel 158,431. Another example if the coordinates x = 2 then in 2 seconds rectangle will shift as much as 7 x 2 x 22,633 = 316,862.

5.2 Testing

Testing is done to determine whether the application has reached the desired goal and weaknesses that have not been resolved in this project so that the future can be done refinement.

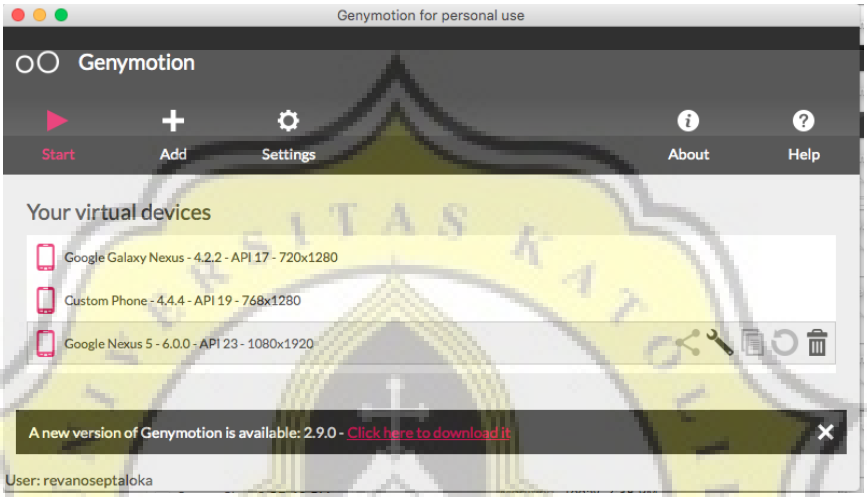


Illustration 5.8: Genymotion Emulator

Testing done on xiaomi with android version 5.1.1, samsung galaxy s7 with android version 7.1.2, emulator genymotion google galaxy nexus with android version 4.4.2 and galaxy nexus 5 with android version 6.0.0.

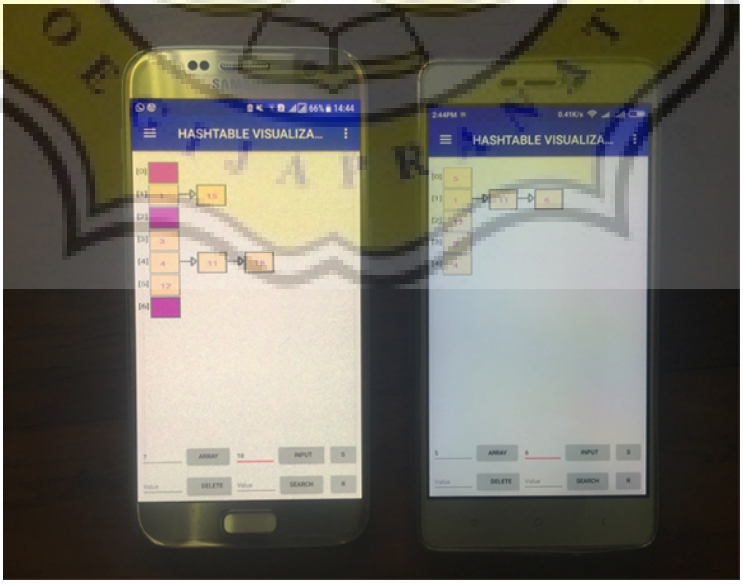


Illustration 5.9: Smartphone Test

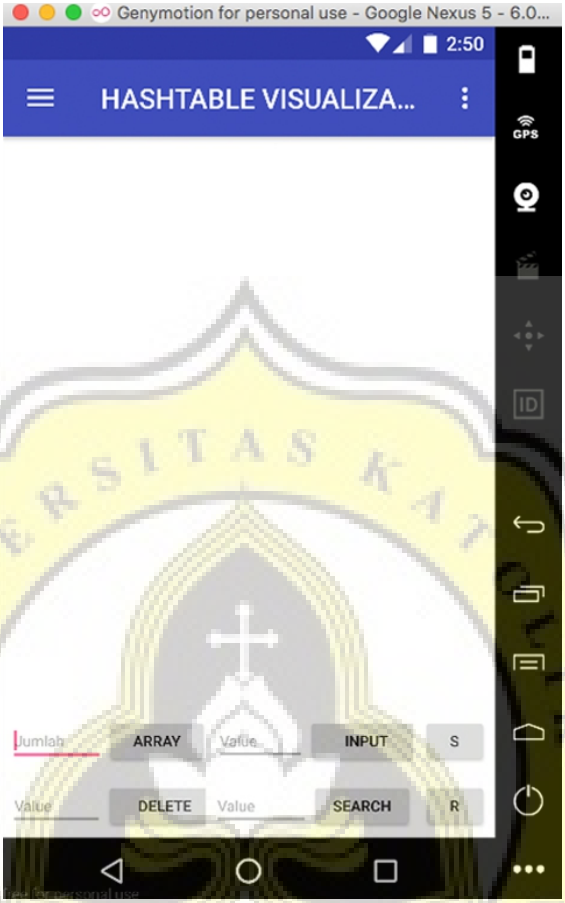


Illustration 5.10: Main Content without Data

From the test on the device the array size, insert, search and delete functions in the process by the hashtable data structure have got the appropriate result.

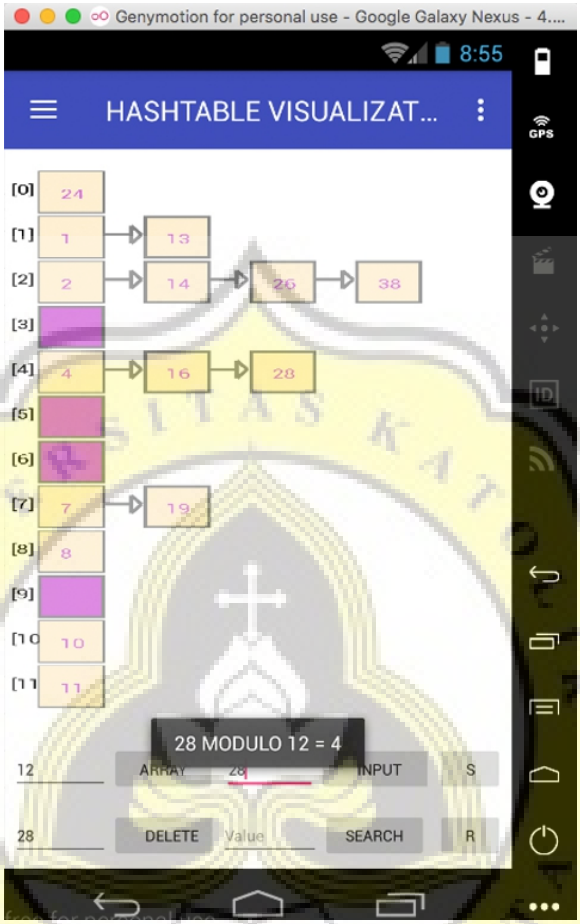


Illustration 5.11: Process of Insert Value

The drawRect process for the insert process is neatly arranged according to the coordinates of the data structure hash table. The shortcomings that still exist in this application is the maximum number of inputan array as many as 12 columns, and list as many as 4 columns.

No.	Input Value	Size of Array	Modulo	Result
1.	24	12	0	TRUE
2.	1	12	1	TRUE
3.	2	12	2	TRUE
4.	4	12	4	TRUE
5.	7	12	7	TRUE
6.	8	12	8	TRUE
7.	10	12	10	TRUE
8.	11	12	11	TRUE
9.	13	12	1	TRUE
10.	14	12	2	TRUE
11.	16	12	4	TRUE
12.	19	12	7	TRUE
13.	26	12	2	TRUE
14.	28	12	4	TRUE
15.	38	12	2	TRUE

Table 5.1: Testing Insert

In the experiments conducted on 15 data, it can be concluded that the data insert process runs well on the data structure and visualization can be displayed precisely according to the coordinates of each value.

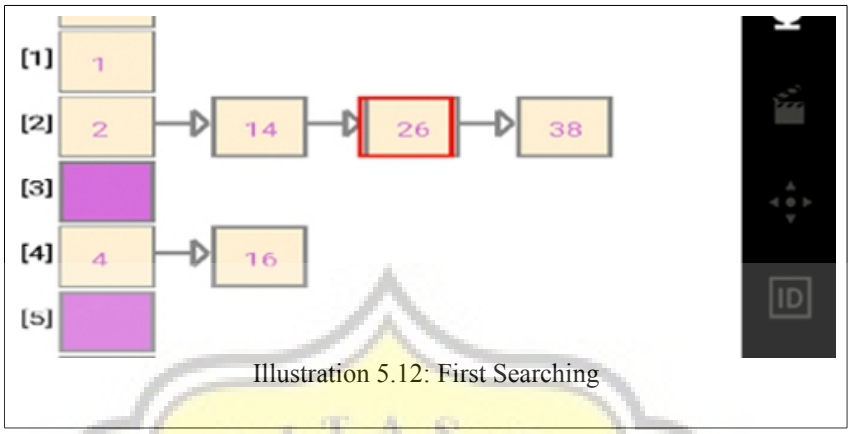


Illustration 5.12: First Searching

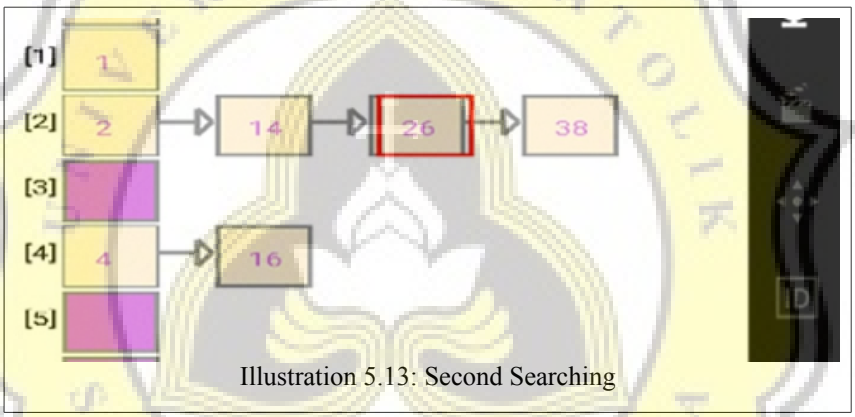


Illustration 5.13: Second Searching

The process of shifting the searching animation has not been maximal, because it is determined by the coundown timer sometimes the rectangle shift goes beyond the coordinates and sometimes has not reached the coordinates, it happens because of the performance of the device. On android devices that have low specifications this will often happen.