令和 6 年 3 月 24 日

京都工芸繊維大学コンピュータ部

Lime 63

はじめに

コンピュータ部部長の石倉です。この度はお忙しい中、我々コンピュータ部にご興味、ご関心を持っていただきありがとうございます。そして、この Lime63 号を見にきてくださったことを、部を代表して厚く御礼申し上げます。この Lime は部員の活動の一部を記したものです。日々の活動の成果や部員個人の趣味など、内容は多岐にわたります。これを通じて、コンピュータ部のことを知っていただき、少しでも我々の活動に興味を持っていただければ幸いです。そして、今回の Lime 作成にあたり記事を執筆した部員各位、ならびに OB・OG の方々を含め、日々我々の活動を支えてくださっているすべての方々への感謝をもって始めの挨拶とさせていただきます。

令和 6 年 3 月 24 日
京都工芸繊維大学コンピュータ部部長　石倉 朋佳

目次

1 C言語における5種類の演算を用いた脳トレ用ツールの開発

情報工学課程 1 回生 田中 大貴

1.1 はじめに

C#を用いて計算問題を解くことができるアプリケーションを作りました。このアプリケーションでは、出題される問題数と問題の難易度を調整して問題を解くことができます。

夏休みまでの C 言語勉強会を終えた時点で Visual Studio Code のターミナルで同様に計算問題の出題、解答を行うことができるプログラムを開発しました。夏休みの時点では、このプログラムの GUI をつくることができる自信がなかったので、春まで寝かせていました。それは、CUI を作成した Visual Studio Code では、C 言語を用いて開発を行っていたが、GUI をつくるソフトウェアでは C#を用いるテンプレートを用いたからです。そこで、夏休みからの半年で C#を少しですが勉強して GUI を作ることができる技能を身につけました。しかし、私の拙い知識では、GUI の完成には漕ぎ着けさせることはできましたが、コードが煩雑だったり、最低限の機能のみの実装に留まったりと、改善が必要なものとなってしまいました。そこで、この記事は私の成長の記録とさせて頂きたく存じます。そのような中でも私の記事に興味を持って頂けた方に厚く御礼を申し上げます。

1.2 概要

このアプリケーションは Visual Studio の Windows Form Application のテンプレートを用いて作成しました。はじめに作成したアプリケーションの概要を紹介させて頂きます。基本的には、出題される問題にテンキーで解答する形式です。また、問題が出題される前に問題数と問題難易度をテンキーから入力することで問題数と難易度を調整できます。

出題される問題について、順序は必ず加法 (足し算)、減法 (引き算)、乗法 (かけ算)、商 (割り算)、剰余 (割り算の余り) の順に出題されます。なお、剰余の次は加法です。また、計算式中に表れる数値 z は全て、乱数で決定されており、出題順が a、難易度が d とする

と、$0 \leqq z \leqq ad$ を満たします。よって、多くの問題に解答したとき、難易度が高く設定されたときは問題に表れる数値が大きくなります。制限時間はなく、チートを検出する機能もないので、無理に暗算しなくとも影響がありません。なお、初版の開発終了後に、0 除算が起こる可能性があることが発覚したため、改善版で修正をしました。また、初版ではマイナスキーを入れていなかったため、減法は全て、大きい方の数から小さい方の数を引く、という形式でした。しかし、第 2 版では、マイナスキーを実装したので、整数の範囲で減法が行われるようになりました。

1.3　ソースコード

知識が拙い故に試行錯誤しながら作成したため、突っ込みを入れたい箇所が多数存在するかとは存じますがご容赦頂けると幸いです。但し、Visual Studio Code で作成した C 言語によるものは省略させて頂きます。ここでは、C#の GUI の第 2 版のソースコードを掲載させて頂きます。

```
1     using System;
2     using System.Collections.Generic;
3     using System.ComponentModel;
4     using System.Data;
5     using System.Drawing;
6     using System.Linq;
7     using System.Text;
8     using System.Threading.Tasks;
9     using System.Windows.Forms;
10
11    namespace Haccason_2nd_2_2
12    {
13        public partial class Form1 : Form
14        {
15            int value = 0, state = 1, correct = 0, clicknum = 0,
                  a = 0, b = 0, diff = 5, num = 10, convert = 0;
16            string answer;
17
18            public Form1()
19            {
20                InitializeComponent();
21                label1.Text = 問題数を入力して下さい".";
22                label2.Text = value.ToString();
23                label3.Text = " ";
24                label4.Text = " ";
25                label5.Text = は、積、"\"*\"\nは、商、\"/\"\nは、割り算の
                      余り剰余\"%\"()を計算して下さい。";
26            }
27
```

```
28              private void button12_Click(object sender, EventArgs
                    e)
29              {
30                  MessageBox.Show入力方法を切り替えました。("");
31              }
32
33              private void radioButton1_CheckedChanged(object
                    sender, EventArgs e)
34              {
35                  convert = 0;
36              }
37
38              private void radioButton2_CheckedChanged(object
                    sender, EventArgs e)
39              {
40                  convert = 1;
41              }
42
43              private void button1_Click(object sender, EventArgs e
                    )
44              {
45                  if(convert == 1)
46                  {
47                      MessageBox.Show現在、あなたはキーボード入力を選択してい
                            ます。キーはご使用頂けません。("10");
48                  }
49                  else
50                  {
51                      value = value * 10 + 1;
52                      label2.Text = value.ToString();
53                  }
54              }
55
56              private void button2_Click(object sender, EventArgs e
                    )
57              {
58                  if (convert == 1)
59                  {
60                      MessageBox.Show現在、あなたはキーボード入力を選択してい
                            ます。キーはご使用頂けません。("10");
61                  }
62                  else
63                  {
64                      value = value * 10 + 2;
65                      label2.Text = value.ToString();
66                  }
67              }
```

```
68
69              private void button3_Click(object sender, EventArgs e
                    )
70              {
71                  if (convert == 1)
72                  {
73                      MessageBox.Show現在、あなたはキーボード入力を選択してい
                            ます。キーはご使用頂けません。("10");
74                  }
75                  else
76                  {
77                      value = value * 10 + 3;
78                      label2.Text = value.ToString();
79                  }
80              }
81
82              private void button4_Click(object sender, EventArgs e
                    )
83              {
84                  if (convert == 1)
85                  {
86                      MessageBox.Show現在、あなたはキーボード入力を選択してい
                            ます。キーはご使用頂けません。("10");
87                  }
88                  else
89                  {
90                      value = value * 10 + 4;
91                      label2.Text = value.ToString();
92                  }
93              }
94
95              private void button5_Click(object sender, EventArgs e
                    )
96              {
97                  if (convert == 1)
98                  {
99                      MessageBox.Show現在、あなたはキーボード入力を選択してい
                            ます。キーはご使用頂けません。("10");
100                 }
101                 else
102                 {
103                     value = value * 10 + 5;
104                     label2.Text = value.ToString();
105                 }
106             }
107
108             private void button6_Click(object sender, EventArgs e
```

```
            )
109         {
110             if (convert == 1)
111             {
112                 MessageBox.Show現在、あなたはキーボード入力を選択してい
                        ます。キーはご使用頂けません。("10");
113             }
114             else
115             {
116                 value = value * 10 + 6;
117                 label2.Text = value.ToString();
118             }
119         }
120
121         private void button7_Click(object sender, EventArgs e
            )
122         {
123             if (convert == 1)
124             {
125                 MessageBox.Show現在、あなたはキーボード入力を選択してい
                        ます。キーはご使用頂けません。("10");
126             }
127             else
128             {
129                 value = value * 10 + 7;
130                 label2.Text = value.ToString();
131             }
132         }
133
134         private void button8_Click(object sender, EventArgs e
            )
135         {
136             if (convert == 1)
137             {
138                 MessageBox.Show現在、あなたはキーボード入力を選択してい
                        ます。キーはご使用頂けません。("10");
139             }
140             else
141             {
142                 value = value * 10 + 8;
143                 label2.Text = value.ToString();
144             }
145         }
146
147         private void button9_Click(object sender, EventArgs e
            )
148         {
```

```
149                 if (convert == 1)
150                 {
151                     MessageBox.Show現在、あなたはキーボード入力を選択してい
                        ます。キーはご使用頂けません。("10");
152                 }
153                 else
154                 {
155                     value = value * 10 + 9;
156                     label2.Text = value.ToString();
157                 }
158             }
159
160             private void button10_Click(object sender, EventArgs
                    e)
161             {
162                 if (convert == 1)
163                 {
164                     MessageBox.Show現在、あなたはキーボード入力を選択してい
                        ます。キーはご使用頂けません。("10");
165                 }
166                 else
167                 {
168                     value = value * 10;
169                     label2.Text = value.ToString();
170                 }
171             }
172
173             private void button11_Click(object sender, EventArgs
                    e)
174             {
175                 if (convert == 1)
176                 {
177                     MessageBox.Show現在、あなたはキーボード入力を選択してい
                        ます。キーはご使用頂けません。("10");
178                 }
179                 else
180                 {
181                     value = - value;
182                     label2.Text = value.ToString();
183                 }
184             }
185             private void button13_Click(object sender, EventArgs
                    e)
186             {
187                 if (convert == 1)
188                 {
189                     MessageBox.Show現在、あなたはキーボード入力を選択してい
```

```
                    ます。キーはご使用頂けません。("10");
190             }
191             else
192             {
193                 value = value / 10;
194                 label2.Text = value.ToString();
195             }
196         }
197
198         private void button14_Click(object sender, EventArgs
                e)
199         {
200             if (convert == 1)
201             {
202                 MessageBox.Show現在、あなたはキーボード入力を選択してい
                        ます。キーはご使用頂けません。("10");
203             }
204             else
205             {
206                 value = 0;
207                 label2.Text = value.ToString();
208             }
209         }
210
211         private void button15_Click(object sender, EventArgs
                e)
212         {
213             clicknum++;
214             if (clicknum == 1)
215             {
216                 if (convert == 1)
217                 {
218                     answer = textBox1.Text;
219                     value = Convert.ToInt32(answer);
220                     textBox1.Text = "";
221                 }
222                 num = value;
223                 label3.Text = 問題数" : " + num;
224                 label1.Text = 問題の難易度を入力して下さい"";
225                 value = 0;
226                 label2.Text = value.ToString();
227             }
228             else if (clicknum == 2)
229             {
230                 if (convert == 1)
231                 {
232                     answer = textBox1.Text;
```

```
233                         value = Convert.ToInt32(answer);
234                         textBox1.Text = "";
235                     }
236                     diff = value;
237                     label4.Text = 難易度" : " + diff;
238                     int seed = Environment.TickCount;
239                     Random rnd = new Random(seed);
240                     a = Convert.ToInt32(rnd.NextDouble() * (state
                            * diff + 1));
241                     Random rand = new Random(seed + a);
242                     b = Convert.ToInt32(rand.NextDouble() * (
                           state * diff + 1));
243                     value = 0;
244                     label2.Text = value.ToString();
245                     label1.Text = state + 問目
                           "" + a + "+" + b + を計算して下さい".";
246                 }
247                 else if (clicknum <= num + 1)
248                 {
249                     if (convert == 1)
250                     {
251                         answer = textBox1.Text;
252                         value = Convert.ToInt32(answer);
253                         textBox1.Text = "";
254                     }
255                     switch (state % 5)
256                     {
257                         case 1:
258                             if (a + b == value)
259                             {
260                                 MessageBox.Show正解です(".");
261                                 correct++;
262                             }
263                             else
264                             {
265                                 MessageBox.Show答えは
                                       ("" + (a + b) + です"");
266                             }
267                             break;
268                         case 2:
269                             if (a - b == value)
270                             {
271                                 MessageBox.Show正解です(".");
272                                 correct++;
273                             }
274                             else
275                             {
```

```
276                             MessageBox.Show答えは
                                    ("" + (a - b) + です"");
277                         }
278                         break;
279                     case 3:
280                         if (a * b == value)
281                         {
282                             MessageBox.Show正解です(".");
283                             correct++;
284                         }
285                         else
286                         {
287                             MessageBox.Show答えは
                                    ("" + (a * b) + です"");
288                         }
289                         break;
290                     case 4:
291                         if (a / b == value)
292                         {
293                             MessageBox.Show正解です(".");
294                             correct++;
295                         }
296                         else
297                         {
298                             MessageBox.Show答えは
                                    ("" + (a / b) + です"");
299                         }
300                         break;
301                     case 0:
302                         if (a % b == value)
303                         {
304                             MessageBox.Show正解です(".");
305                             correct++;
306                         }
307                         else
308                         {
309                             MessageBox.Show答えは
                                    ("" + (a % b) + です"");
310                         }
311                         break;
312                 }
313                 state++;
314                 value = 0;
315                 label2.Text = value.ToString();
316                 int seed = Environment.TickCount;
317                 Random rnd = new Random(seed);
318                 a = Convert.ToInt32(rnd.NextDouble() * (state
                        * diff + 1));
```

```
319                     Random rand = new Random(seed + a);
320                     b = Convert.ToInt32(rand.NextDouble() * (
                            state * diff + 1));
321                     switch (state % 5)
322                     {
323                         case 1:
324                             label1.Text = state + 問題
                                ""+ a + "+" + b + を計算して下さい
                                ".";
325                             break;
326                         case 2:
327                             label1.Text = state + 問題
                                ""+ a + "-" + b + を計算して下さい
                                ".";
328                             break;
329                         case 3:
330                             label1.Text = state + 問題
                                ""+ a + "*" + b + を計算して下さい
                                ".";
331                             break;
332                         case 4:
333                             int count1 = 1;
334                             while(b == 0)
335                             {
336                                 count1++;
337                                 Random rand1 = new Random(seed +
                                    a * count1);
338                                 b = Convert.ToInt32(rand.
                                    NextDouble() * (state * diff
                                    + 1));
339                             }
340                             label1.Text = state + 問題
                                ""+ a + "/" + b + を計算して下さい
                                ".";
341                             break;
342                         case 0:
343                             int count2 = 1;
344                             while (b == 0)
345                             {
346                                 count2++;
347                                 Random rand1 = new Random(seed +
                                    a * count2);
348                                 b = Convert.ToInt32(rand.
                                    NextDouble() * (state * diff
                                    + 1));
349                             }
350                             label1.Text = state + 問題
                                ""+ a + "%" + b + を計算して下さい
                                ".";
351                             break;
```

```
352                     }
353                 }
354                 else
355                 {
356                     switch (state % 5)
357                     {
358                         case 1:
359                             if (a + b == value)
360                             {
361                                 MessageBox.Show正解です(".");
362                                 correct++;
363                             }
364                             else
365                             {
366                                 MessageBox.Show答えは
                                        ("" + (a + b) + です"");
367                             }
368                             break;
369                         case 2:
370                             if (a - b == value)
371                             {
372                                 MessageBox.Show正解です(".");
373                                 correct++;
374                             }
375                             else
376                             {
377                                 MessageBox.Show答えは
                                        ("" + (a - b) + です"");
378                             }
379                             break;
380                         case 3:
381                             if (a * b == value)
382                             {
383                                 MessageBox.Show正解です(".");
384                                 correct++;
385                             }
386                             else
387                             {
388                                 MessageBox.Show答えは
                                        ("" + (a * b) + です"");
389                             }
390                             break;
391                         case 4:
392                             if (a / b == value)
393                             {
394                                 MessageBox.Show正解です(".");
395                                 correct++;
```

```
396                                 }
397                                 else
398                                 {
399                                     MessageBox.Show答えは
                                            ("" + (a / b) + です"");
400                                 }
401                                 break;
402                             case 0:
403                                 if (a % b == value)
404                                 {
405                                     MessageBox.Show正解です(".");
406                                     correct++;
407                                 }
408                                 else
409                                 {
410                                     MessageBox.Show答えは
                                            ("" + (a % b) + です"");
411                                 }
412                                 break;
413                         }
414                         state++;
415                         value = 0;
416                         label2.Text = value.ToString();
417                         label1.Text = あなたは" " + num + " 問のうち
                                " + correct + " 問正解されました.";
418                     }
419                 }
420             }
421     }
```

ソースコード 1.1　calculation_training.cs

1.4　実行時の画面

以下では、基本的には初版の画面を示します。第 2 版の画面は後に示します。以下に 1.1 として示すものは、開始時の画面です。

実際に実行します。はじめに、問題数を入力するように指示があるので、問題数をテンキーで入力して頂けると幸いです。以下に 1.2 として示すものは、問題数入力の例です。ここでは、10 問と入力しました。

次に、難易度を入力するように指示されますので、難易度をテンキーで入力して頂けると幸いです。以下に難易度設定の入力の例を 1.3 として示します。ここでは難易度を 5 に設定しました。

難易度選択が終了すると、問題が表示されます。問題表示の例を 1.4 として以下に示します。

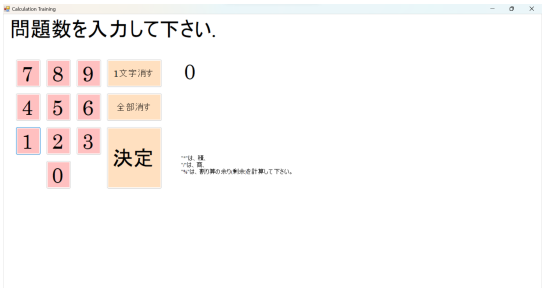

図 1.1　開始時の画面です

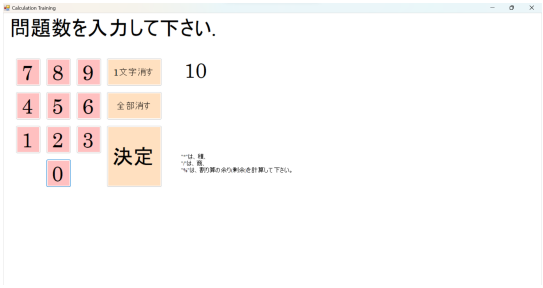

図 1.2　問題数入力の一例です

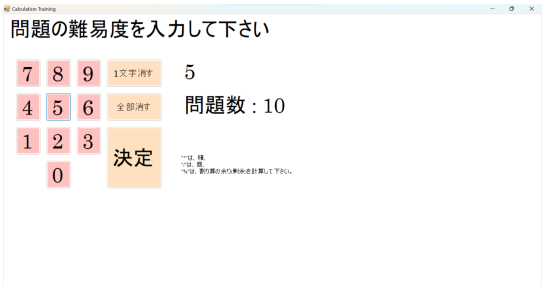

図 1.3　難易度入力の一例です

正誤判定された後の画面の例を 1.5, 1.6 として以下に示します。

指定した問題数分の問題を解ききると、○問のうち△問正解、のように出力されます。このときの画面の例を 1.7 として以下に示します。

ここからは、第 2 版で改変を加えた部分について言及します。第 2 版での開始時の画面は以下に 1.8 として示します。

1. マイナスキーの実装
2. キーボード入力機能の実装
3. キーボード入力と 10 キーの切り替え機能の実装

の変更を加えました。

マイナスキーについては、全ての整数 a, b に対して差 $a - b$ を計算できるように実装しました。マイナスキーを使用する際は、差の絶対値を入力してからマイナスキーを押して

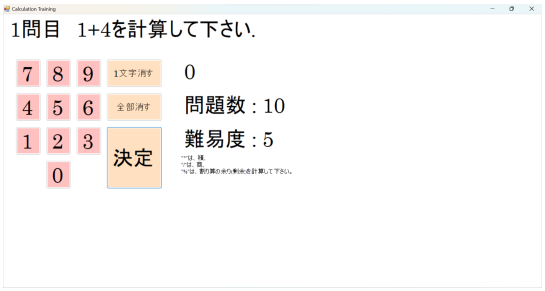

図 1.4　問題画面の一例です

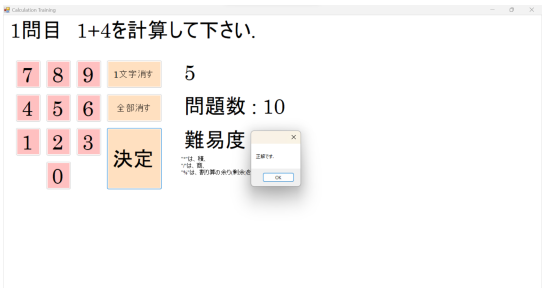

図 1.5　正解したときの画面の一例です

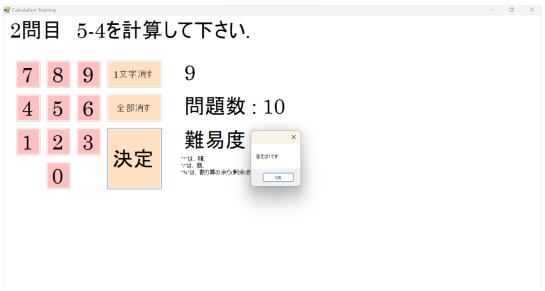

図 1.6　不正解のときの画面の一例です

下さい。例えば、−5 と答えたいときは、5, − の順に押して下さい。

キーボード入力機能については、キーボードから解答を行いたい場合を想定して実装しました。入力する際は、10 キーの上のグレーの四角形の部分が入力スペースになっていますので、グレーの四角の部分をクリックして入力して下さい。

キーボード入力と 10 キーの切り替え機能については、ラジオボタンを用いて制御しています。1.8 では、10 キーを選択しています。しかし、キーボードを選択すれば、キーボード入力に切り替わるとともに、10 キーの入力を受け付けなくなります。

キーボードに切り替えるときは、1.9 のように「キーボード」を選択し、下の「切り替え」を押せば、キーボード入力に切り替わります。1.10 のように表示されれば切り替えは完了です。 なお、10 キーへの切り替えのときは 10 キーを選び、「切り替え」を押せばよいです。

ここで、キーボード入力に切り替えたときに 10 キー入力を行うことを考えます。この

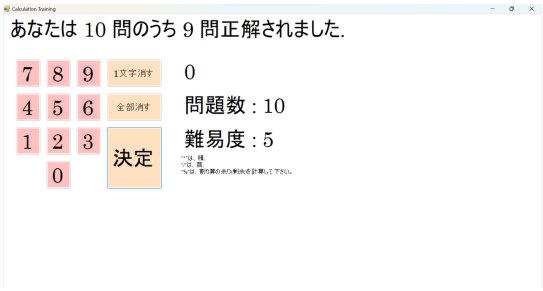

図 1.7　結果画面の一例です

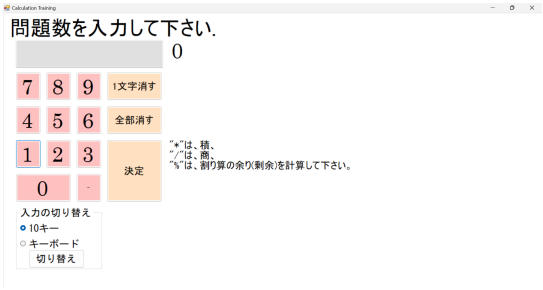

図 1.8　第 2 版での開始時の画面です

図 1.9　キーボード入力への切り替え方です。

とき、1.11 のように出力されて、入力が受け付けられません。

実際に入力すると、1.12 のようになります。

また、10 キー入力の状態でキーボードから入力し、解答を確定すると、1.13 のように、0 と解答したものとして正誤判定を行います。

1.5　おわりに

私の記事をご通読頂き、ありがとうございました。C 言語、C#を用いることで、素人の私でも簡単なアプリケーションを開発することができることに驚かされました。この記

図 1.10　このように出力されたら切り替えは完了です。

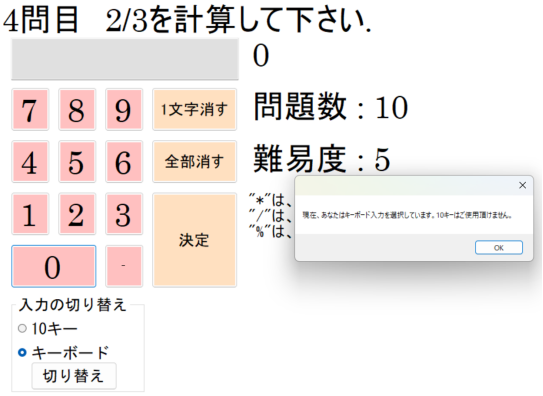

図 1.11　このように出力されたら切り替えは完了です。

図 1.12　キーボード入力の一例です。

事の中にはお見苦しい点も多々あるかとは思いますが、ご容赦頂けると幸いです。作成する中で、調べたことは少しずつでも覚えていこうと思いました。素人が調べながら試行錯誤して作成したアプリケーションなので、違和感を抱かれる点が多々あるかとは思います

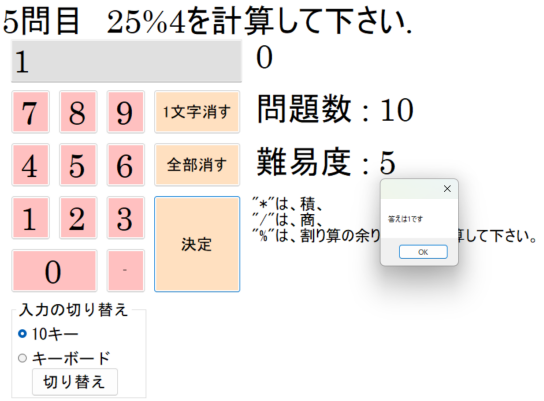

図 1.13　キーボード入力の一例です。

が、そっと心の中にしまっておいて頂けるとありがたいです。

参考文献

[1] 文字列の読み取りについて C#テキストボックスに入力された文字列を取得する|iPentec

[2] 乱数の生成について【C#,VB.NET】乱数の生成#C#-Qiita

[3] 乱数の生成について C#Randomize の処理をしたい|iPentec

[4] 乱数の生成について [C#]Random の NextDouble() で特定の範囲の小数を生成する (generatedoublenumberinrange)には？|ちょげぶろぐ (choge-blog.com)

[5] double 型から int 型への変換について C#で Double を Int に変換する|Delft スタック (delftstack.com)

[6] 乱数のシード値の変更について C#同じ値が返る？ Random クラスを利用して 6 桁整数の乱数発生させる|オレンジの国 (techlive.tokyo)

[7] .exe ファイルの作成について windows アプリケーションを exe ファイルとしてビルドする方法（VB.NET）#VisualStudio-Qiita

[8] ラジオボタンの実装について C#ラジオボタンの作成のサンプル (フォームアプリケーション)|ITSakura

[9] lime の記事の書き方の参考にしました http://www.kitcc.org/

2 スタンプカードアプリの開発

応用化学課程 2 回生 大西晴太

今年の春休みにハッカソンが行われ、私は石倉さん、樋口さんと共にアンドロイドアプリの開発を試みた。以下、私はアプリの概要のみを記すが機能やコードの詳細については石倉さん、樋口さんの記事を参照されたい。

2.1 開発の背景

去年の夏休みのハッカソンで「大学生活に役立つアプリ」の開発を試みたが、多機能なものにしようとしたこともあり、ハッカソンの期間中に完成させることができなかった。このことと今回のハッカソンでは前回に比べて製作期間が短かいことを考慮し、デザインを含め、完成品を開発することを主な目標として開発に挑んだ。今回も前回と同様に、「大学生活に役立つアプリ」を開発することにした。そして、大学生は日々、バイトやレポート課題など様々な予定を組んで生活していることを考慮し、ToDo リストの機能を持つアプリを開発することとなった。

2.2 アプリの概要

このアプリはやりたいことやタスクをリストに入力し、達成できたらスタンプを押すことができるという、ToDo リストとスタンプカードを融合したようなアプリである。そして、リストの数は自由に増減可能で、各 ToDo リストをグループ化し、タイトルを付けることができ、また、アプリの背景及びスタンプを様々なデザインに変更できるという特長がある。各デザインにおける使用例を以下に示す。

2.3 終わりに

今の世の中では、ほとんどの人がスマートフォンを所持し、日々、様々なアプリを用いており、アプリがなければ今の我々の生活は成り立たないと言えるだろう。私はそのような生活必需品とも言えるアプリの開発段階の一端を前回、そして今回のハッカソンで学習してきたが、勉強すればする程、奥が深いことがわかり、より一層、アプリ開発に興味を

図 2.1　各デザインにおける使用例

持った。私はこれからもアプリについての勉強を継続し、ゆくゆくは今までよりも多機能で、かつ実用的で製品化され得るようなアプリを開発したい。

3 スタンプカードアプリ

電子システム工学課程 1 回生 樋口舞子

3.1 はじめに

今回、グループ開発でスタンプカードアプリを作成した。この記事では、計画と実行、そして私が担当した Room・見た目・入力機能の実装について述べていく。

3.2 計画と実行

3.2.1 計画

discord を使って事前準備を行い、ハッカソン当日に完成させるという流れで開発を進めていった。開発環境として android studio を使い、言語は Kotlin を使用した。事前準備は、私と大西さんが実装を担当し、石倉さんが私たち 2 人のサポートを担当することになった。

第 1 回　見た目の作成
第 2 回　スタンプ機能の実装、値保持機能の導入
第 3 回　第 2 回の続き、値保持機能の実装完成、追加機能
ハッカソン 1 日目　細部・追加機能、テストなど
ハッカソン 2 日目　細部・追加機能、テストなど、発表準備

3.2.2 実行

実際には次のような流れで計画を進めた。

- 第 1 回
  - アプリの概要などの共有
  - 見た目の作成
  - 入力機能を実装

- 第 2 回
  - スタンプ機能の実装
  - 値保持機能（Room のみ）の実装
- 第 3 回
  - 値保持機能（Room のみ）ViewModel の作成
  - 時間外のタスク
    - 値保持機能（DataStore）の実装
    - 追加機能：柄変更機能の実装
    - 追加機能、アプリのアイコンやスタンプの画像の案を出す
- ハッカソン 1 日目
  - APK ファイルの生成方法の確認
  - 実機での動作の確認
  - アイコンの決定
  - 追加機能: 項目全削除機能の実装
  - 時間外のタスク
    - アプリアイコンの実装
    - カード追加機能の実装の途中まで
    - 発表スライドの作成
- ハッカソン 2 日目
  - カード追加・変更機能の実装
  - 項目追加機能の実装
  - 項目削除機能を追加
  - 最終版の APK ファイルを生成（Release 版でない）

3.3 コード解説

Room・見た目・入力機能の実装コードの詳細について述べていく。

3.3.1 見た目

Image 関数

Image 関数は画像を出力するための関数である。画像を android studio 上にダウンロードし、Image 関数を使い表示させた。ソースコード 5.1 は画像ダウンロード後のプログラムの例である。画像を表示させる機能だけしかもたない最も簡素な形を表している。

```
1     @Composable
2     Image(
3         painter = painterResource(R.drawable.ic_task_completed)
```

```
4           // 画像ファイル名はic_task_completed
5           contentDescription = null
6       )
```

ソースコード 3.1 Image 関数

Column 関数

関数を縦に並べる

Modifier

見た目の設定を変更できる。

- .weight
  幅の指定
- .padding
  余白の指定ができる。
- .fillMaxWidth()
  画像を画面の最大幅で表示させる

3.3.2 入力機能

TextFIeld 関数

TextField 関数はテキストボックスを出力するための関数である。

```
1       @OptIn(ExperimentalMaterial3Api::class)
2       @Composable
3       var sentence by remember { mutableStateOf( "" )}
4       // 変数の初期化sentence
5       TextField(
6               value = sentence,
7               onValueChange = {sentence = it},
8               // キーボード入力で文字表示
9               modifier = Modifier
10                  .weight(4f)
11                  .padding(8.dp),
12              singleLine = false,
13              keyboardOptions = KeyboardOptions(keyboardType =
                    KeyboardType.Text, imeAction = ImeAction.Done)
14              // 決定ボタンに反応、テキストキーボードに対応
15          )
```

ソースコード 3.2 TextField 関数

3.3.3 Room

Room とは、データベース操作を簡素化する値保持機能の 1 つである。Room は次の手順で実装する。

1. Room の依存関係を追加する
2. アイテムエンティティを作成する
   Entity クラスはテーブルを定義する。項目 1 つがテーブルの行に相当し、項目が持つ属性が列に相当する。
   - com.example.stampcard 直下に、data パッケージを作成。
   - data 直下に、room パッケージを作成
   - room 直下に、item パッケージを作成。
   - data.room.item 直下に、Item クラスを作成。

```
1     @Entity(tableName = "items")
2     data class Item (
3         @PrimaryKey(autoGenerate = true)
4         val id: Int = 0,
5         val name: String,
6         val checked: Boolean,
7         val card: Int = 0
8         )
```

ソースコード 3.3 Item クラス

3. アイテム DAO を作成する
   データアクセスオブジェクト（DAO） は、データベース操作に関連する部分をアプリの残り部分から分離させ、データベースの照会、取得、挿入、削除、更新を簡単に行うことができる。Room はコンパイル時にこのクラスの実装を生成する。data.room.item 直下に、ItemDao インタフェースを作成。ソースコードは図 3.1。

```
@Dao
interface ItemDao {

@Insert(onConflict = OnConflictStrategy.IGNORE)
suspend fun insert(item: Item)
// 挿入機能

@Update
suspend fun update(item: Item)
// 更新機能

@Delete
suspend fun delete(item: Item)
// 削除機能

@Query("SELECT * from items WHERE id = :id")
// itemsから、idが:card引数と一致する列がすべて選択される
fun getItem(id: Int): Flow<Item>

@Query("SELECT * from items ORDER BY id ASC")
// id順にitemを照会
fun getAllItems(): Flow<List<Item>>

@Query("SELECT * from items WHERE card = :card")
//itemsから、cardが:card引数と一致する列がすべて選択される
fun getCardItemStreams(card: Int): Flow<List<Item>>
}
```

図 3.1 DAO

4. データベースインスタンスを作成する
   データベースインスタンスとは Entity と DAO の処理を実行するオブジェクトである。data.room 直下に、StampcardDatabase インタフェースを作成。

```
1   @Database(entities = [Item::class, Card::class], version
        =1, exportSchema = false)
2   abstract class StampcardDatabase : RoomDatabase() {
3   abstract fun itemDao(): ItemDao
4   abstract fun cardDao(): CardDao
5   companion object{
6       @Volatile
7       private var Instance: StampcardDatabase? = null
8       fun getDatabase(context: Context): StampcardDatabase
            {
9           return Instance ?: synchronized(this){
10              Room.databaseBuilder(context,
                    StampcardDatabase::class.java, "
                    item_database")
11                  .build()
12                  .also { Instance = it }
13                  }
14          }
15      }
```

```
16          }
```

ソースコード 3.4 データベースインスタンス

5. リポジトリを実装する
   リポジトリとは、ファイルやプログラム、設定情報などを保管する場所のことである。ここでは、値保持機能を実装する。
   - data.room.item 直下に、ItemsRepository インターフェースを作成。

```
1           interface ItemsRepository {
2               fun getAllItemsStream(): Flow<List<Item>>
3               fun getItemStream(id: Int): Flow<Item?>
4               fun getCardItemStreams(card: Int): Flow<List<
                    Item>>
5               fun deleteAllItemsOfCard(card: Int): Unit
6               suspend fun insertItem(item: Item)
7               suspend fun deleteItem(item: Item)
8               suspend fun updateItem(item: Item)
9               }
```

ソースコード 3.5 ItemsRepository

   - data.room.item 直下に、 OfflineItemsRepository クラスを作成。

```
class OfflineItemsRepository(private val itemDao: ItemDao) : ItemsRepository {
    override fun getAllItemsStream(): Flow<List<Item>> = itemDao.getAllItems()

    override fun getItemStream(id: Int): Flow<Item?> = itemDao.getItem(id)

    override fun getCardItemStreams(card: Int): Flow<List<Item>> = itemDao.getCardItemStreams(card)

    override fun deleteAllItemsOfCard(card: Int) = itemDao.deleteAllItemsOfCard(card)

    override suspend fun insertItem(item: Item) = itemDao.insert(item)

    override suspend fun deleteItem(item: Item) = itemDao.delete(item)

    override suspend fun updateItem(item: Item) = itemDao.update(item)
}
```

図 3.2 OfflineItemsRepository

   - data.room 直下に、AppContainer.kt を作成する。
   - com.example.stampcard 直下に StampcardApplication クラスを作成

6. ViewModel を作成し、機能を追加する
   - AppViewModelProvider の作成
   - StampcardViewModel の作成
   - StampcardScreen への変更

```
interface AppContainer {
    val itemsRepository: ItemsRepository
    val cardsRepository: CardsRepository
}


class AppDataContainer(private val context: Context) : AppContainer {
    override val itemsRepository: ItemsRepository by lazy {
        OfflineItemsRepository(StampcardDatabase.getDatabase(context).itemDao())
    }
    override val cardsRepository: CardsRepository by lazy {
        OfflineCardsRepository(StampcardDatabase.getDatabase(context).cardDao())
    }
}
```

図 3.3　AppContainer

```
class StampcardApplication : Application() {
    lateinit var container: AppContainer
    lateinit var userPreferencesRepository: UserPreferencesRepository

    override fun onCreate() {
        super.onCreate()
        container = AppDataContainer( context: this)
        userPreferencesRepository = UserPreferencesRepository(dataStore)
    }
}

private const val TITLE_PREFERENCE_NAME = "title_preferences"
private val Context.dataStore: DataStore<Preferences> by preferencesDataStore(
    name = TITLE_PREFERENCE_NAME
)
```

図 3.4　StampcardApplication

3.4　さいごに

プログラミング初心者ではあったが、石倉さんの手厚いサポートのおかげもあり、一部機能の実装をすることができた。アプリが実機で実行される姿を見たときは本当に嬉しかった。しかし、公式チュートリアルを読んでも理解できない部分はまだまだ多い。今後、勉強して個人開発をできるような力をつけていきたいと考えている。

```
object AppViewModelProvider {
    val Factory = viewModelFactory { this: InitializerViewModelFactoryBuilder
        initializer { this: CreationExtras
            StampcardViewModel(
                stampcardApplication().container.itemsRepository
            )
        }
    }
}


fun CreationExtras.stampcardApplication(): StampcardApplication =
    (this[ViewModelProvider.AndroidViewModelFactory.APPLICATION_KEY] as StampcardApplication)
```

図 3.5　AppViewModelProvider

```
1  data class HomeUiState(val itemList: List<Item> = listOf())
2  class StampcardViewModel (private val itemsRepository: ItemsRepository) : ViewModel() {
3      companion object {
4          private const val TIMEOUT_MILLIS = 5_000L
5      }
6
7      val homeUiState: StateFlow<HomeUiState> =
8          itemsRepository.getAllItemsStream().map { HomeUiState(it) }
9              .stateIn(
10                 scope = viewModelScope,
11                 started = SharingStarted.WhileSubscribed(TIMEOUT_MILLIS),
12                 initialValue = HomeUiState()
13             )
14     private suspend fun insertItem(item: Item = Item(name = "", checked = false)) {
15         itemsRepository.insertItem(item)
16     }
17     suspend fun updateItem(item: Item) {
18         itemsRepository.updateItem(item)
19     }
20
21     suspend fun initialize() {
22         if(homeUiState.value.itemList.isEmpty()) {
23             for(count in 1 .. 5) {
24                 insertItem()
25             }
26         }
27     }
28 }
```

図 3.6　StampcardViewModel

参考文献

[1] https://developer.android.com/courses/android-basics-compose/course?hl=ja

[2] https://kitcc.esa.io/posts/66

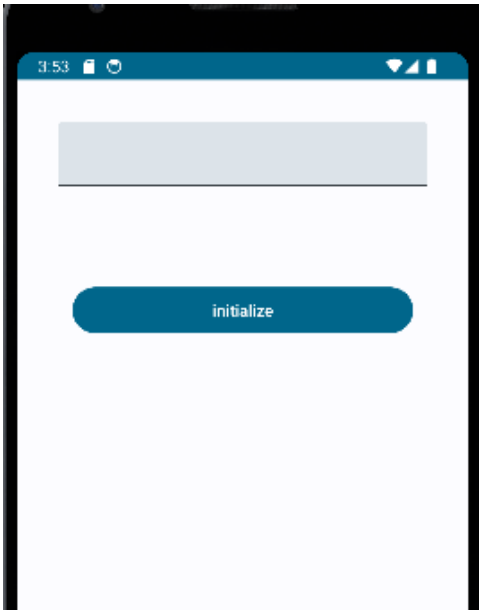

図 3.7　Room 実装後の画面 1

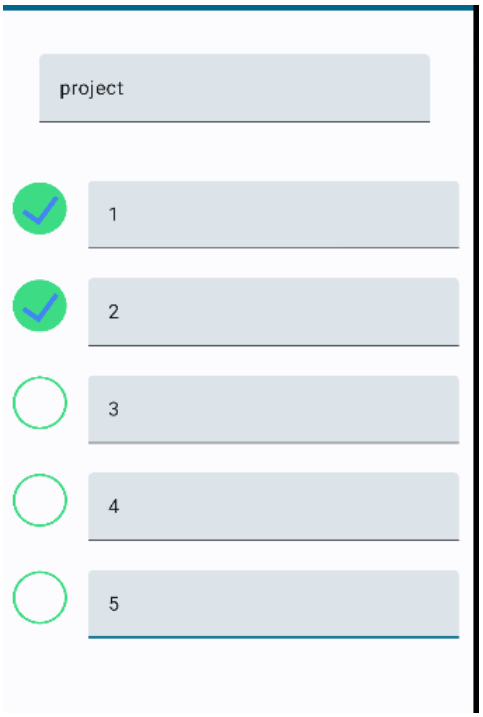

図 3.8　Room 実装後の画面 2

4 スタンプカードアプリの構想から実装

情報工学課程 2 回生 石倉 朋佳

4.1 はじめに

Android Studio と Kotlin を使って，グループでアプリ開発をしました．この春に開発した「スタンプカードアプリ」は，やることをいくつか書いておいて，達成した項目にはスタンプを押す，TODO リストのように使えるアプリです．

この記事では，このスタンプカードアプリの一部の機能について，構想から実装までどのように考えて開発を進めたかを書きます．春のアプリ開発では，明確に設計担当などを決めたわけではありませんが，私は主に構想から設計のところを担いました．他のメンバーに実装してもらうにあたり，設計や実装の方法は共有しましたが，構想からどのように決めたのかという話までする機会はあまり無かったため，記事にしようと思います．また，実際にその機能を実装したコードも一部紹介したいと思います．

以降の節では，スタンプカードアプリの機能のうち，スタンプ機能，値保持機能，柄変更機能の 3 つについて説明します．

4.2 スタンプ機能

まず，「スタンプ機能」の説明とその実装までの流れについて説明します．

機能の説明

図 4.1 にスタンプ機能の説明を示します．左の画面の「スタンプの枠」の画像をタップすると，右の画面のようにスタンプが押された画面になります．また，右の画面でスタンプの画像をタップすると左の画面になるため，一度押したスタンプを取り消すこともできます．

構想から実装

スタンプカードアプリは，やることを達成したらスタンプを押すことができるというアプリです．そのため，画面にはスタンプを押す前の枠が表示されているときと，押されたスタンプが表示されているときがあり，これをユーザの意思で切り替えられるようにしたいと思いました．

図 4.1　スタンプ機能の説明

ここで，Android Studio 公式のチュートリアル「Compose を用いた Android アプリ開発の基礎」コース 内で学んだことが使えると思いました，特に，DiceRoller アプリを作る中で使った方法が使えると思いました．公式チュートリアルのユニット 2 パスウェイ 3「UI と状態を操作する」で学んだ、画像をボタンにする方法や，ボタンを押すと画像のリソースを切り変える方法，アプリの実行中に状態を保持する方法を使います． [1]

スタンプカードアプリでは，これを「スタンプの枠」の画像と「スタンプ」の画像を切り替えることで，スタンプ機能として実装しました．公式チュートリアルの該当ページを見直し，大体以下の流れで実装できると考えました．

- 状態変数を用意する
  - checked 変数：「スタンプが押されているかどうか」を格納する Boolean 型の状態変数．
- 状態変数の値によって画像が変わるようにする
  - checkimage 変数：画像のコンポーザブルに渡す，リソースを格納する変数．checked 変数の値によって「スタンプの枠」の画像になるか「スタンプ」の画像になるかが決まる．
- 画像をタップすることで状態変数の値が切り替わるようにする
  - 画像のコンポーザブル Image の修飾子に .clickable を付ける．
  - クリックするときに実行してほしい関数を clickable の引数 onClick に渡す．ここでは，Boolean 型の変数 checked の値を切り替えたいため， checked に checked の否定を代入する．

このようにして実装したものを，ソースコード 4.1 に示します．これはアプリが完成したときの最終版のコードではなく実装途中のときものですが，スタンプ機能を実装した直後で分かりやすいためこちらを載せます．

```
1     @OptIn(ExperimentalMaterial3Api::class)
2     @Composable
3     fun CheckandImage(){
4         var sentence by remember { mutableStateOf("") }
5         var checked by remember { mutableStateOf(false) }
6         val checkimage = if (checked) painterResource(id = R.
              drawable.ic_task_completed) else painterResource(id =
               R.drawable.ic_task_unfinished)
7
8         Row(modifier = Modifier.padding(4.dp)){
9             Image(
10                painter = checkimage,
11                contentDescription = null,
12                modifier = Modifier
13                    .weight(1f)
14                    .padding(8.dp)
15                    .clickable(onClick = { checked = !checked })
16            )
17            TextField(
18                value = sentence,
19                onValueChange = { sentence = it },
20                modifier = Modifier
21                    .weight(5f)
22                    .padding(8.dp)
23            )
24        }
25    }
```

ソースコード 4.1 スタンプ機能の実装

画像をアプリの画面に表示するとき，画面を構成するコンポーザブルの中で Image コンポーザブルを呼び出します．表示したい画像をリソースとしてプロジェクトに事前にインポートして，その ID を Image コンポーザブルに渡すことで，スタンプやその枠の画像を表示することができます．

4.3 値保持機能

次に，「値保持機能」の説明とその構想について説明します．

機能の説明

スタンプ機能を実装しただけでは，アプリを閉じるとテキストフィールドに入力した文字やスタンプの状態の情報が失われてしまい，毎回アプリを開くたびに初期化されてしまいます．そのため，それらの値を保持する機能を実装する必要があります．これをここでは「値保持機能」と呼び，Room を用いたデータベースや DataStore を使って実装しま

した.

機能の構想

Room を用いると, 関係データベースと SQL を利用できるため, やることを書くテキストフィールドと画像の状態を保存するのにちょうどよいと思いました．一方，DataStore を用いた値の保存はデータが 1 つだけのときのほうが適してるため，開発当初はカードのタイトルを入力するテキストフィールドの値を保存するために実装しました．これは後に追加機能を実装したときに使わなくなりましたが，代わりに保持すべき値も増えたため，その保持に使っています.

DataStore を用いた値保持機能は公式チュートリアルの該当ページに倣って実装したため，実装したコードなどは省きます．実装の流れは以下の通りです.

- 依存関係の設定
- ユーザー設定リポジトリの実装
- DataSotre の初期化
- ViewModel の実装
- 画面のコンポーズ関数への変更
  公式チュートリアル [1] ユニット 6 パスウェイ 3(3)「DataStore を使用して設定をローカルに保存する」を参考に作成

最初，このアプリのプロトタイプを作ったときは，スタンプ機能さえ実装すれば思ったようなアプリができると思っていました．しかし，状態変数に remember 関数を用いても，値が保持されるのはアプリが実行している間だけであり，一度アプリを閉じるとその情報は失われてしまいます．値保持機能は構想段階では気づかないが実は必要な機能であり，この機能も含めて実装することで初めて，本来期待していた通りの振る舞いをするようになりました.

4.4 柄変更機能

最後に,「柄変更機能」の説明とその実装までの流れについて説明します.

機能の説明

「柄変更機能」とは，アプリのスタンプや背景の見た目を変える（既存のセットから選べる）機能です．図 4.3 において柄変更機能の説明をします．左の画面について，指で指しているこのアイコンをタップすると，中央の画面のようにドロップダウンメニューが展開します．「footprint」が選択されていましたが，右の画面のように「sakura」を選択すると，スタンプと背景の柄が変わります.

最初の構想

画像の切り替えはスタンプ機能のときと同様にできるため，大体以下の流れで実装できるだろうと考えました.

図 4.2　柄変更機能の説明

- 背景を導入する
  - Box コンポーザブルを用いて背景画像を UI の下に重ね，背景画像を反映させる．
- 状態変数を用意する
  - currentPattern 変数：現在の柄を表す整数を格納する状態変数．
- 状態変数の値によって画像が変わるようにする
  - スタンプの画像のリソースを格納する checkimage 変数に入れる値を改める．when 式を使い，状態変数の値によって異なる「スタンプの枠」または「スタンプ」の画像が表示されるようにする．（C 言語であれば条件分岐を switch 文を使って書くのに近い）
  - 背景画像のリソースを格納する変数を作り，状態変数の値によって異なるリソースが格納されるようにする．
- 柄を変更するための UI を実装する

最初は，「状態変数の値によって画像が変わるようにする」ために when 式の条件分岐を用いたため，ソースコード 4.2 のようなコードを 2 か所に書きました．また，図 4.3 に示すドロップダウンメニューから柄を選択するため，柄の名前についても同様に画面表示に関する関数内に書いていました．しかし，それによって柄を 1 種類増やすときに 3 か所以上コードを書き換える必要があり，柄を追加するときのコードの変更ミスによりアプリが動かなくなることがありました．

```
1       val currentPattern by remember { mutableStateOf(0) }
2       val checked by remember { mutableStateOf(false) }
3
```

```
4       val resourceID = when(currentPattern) {
5           1 -> if (checked) R.drawable.default_checked
6                   else R.drawable.default_unchecked
7           2 -> if (checked) R.drawable.footprint_checked
8                   else R.drawable.footprint_unchecked
9           3 -> if (checked) R.drawable.good_checked
10                  else R.drawable.dood_unchecked
11          // 開発者が柄を追加したいとき，ここや他のところにも書き加える必要がある
12          else -> if (checked) R.drawable.sakura_checked
13                  else R.drawable.sakura_unchecked
14      }
15
16      Image(
17          painter = painterResource(id = resourceID),
18          contentDescription = null,
19          modifier = Modifier
20              .weight(1f)
21              .padding(8.dp)
22              .clickable(onClick = { checked = !checked })
23      )
```

ソースコード 4.2　when 式を使って柄変更機能を実装する場合

採用した構想・実装

そこで，柄のためのデータを新たに定義することに決めました．大体以下のような流れで実装しました．UI 状態を保持するための実装には，参考文献 [2] および [3]，[4] を参考にして実装しました．

- 柄のためのデータの定義
- 柄を変更するための UI の実装
- UI 状態を保持するための実装
- 画面のコンポーズ関数への変更

柄変更機能を実装するにあたり，「柄の名前」，「スタンプの枠の画像」，「スタンプを押したときの画像」，「背景画像」という 4 種類のデータを暗に 1 つのセットとして扱うようになっていました．これをコードの別々のところで扱うとミスとバグのもとになるため，これらをまとめて Pattern クラスとしました．Pattern クラスの定義をソースコード 4.3 に示します．Pattern クラスは前述した 4 つのプロパティをもつだけのクラスなので， Room によるデータベースの Entity と同様にデータクラスにしています．

```
1       data class Pattern(
2           val name: String,
3           val backgroundId: Int?,
4           val checkedId: Int,
5           val uncheckedId: Int
```

```
6       )
```

ソースコード 4.3 柄変更機能に用いるクラス Pattern

データベースに保存するようなデータと違い，柄変更機能に用いる柄のデータはアプリのユーザが変更することはありません．そのため，ソースコード 4.4 のように，ローカルデータとして PatternData オブジェクトをファイルに書いておくことにしました．arrayOf() 関数を用いて配列で保存しておくことで，配列の添え字で何番目の柄であるかを指定でき，柄の種類の数を.size() で取得することができるようになりした．これにより，画面を構成するコンポーズ関数の中にマジックナンバーなどを書く必要が無くなりました．開発メンバーが柄をアプリに追加したいときは，プロジェクトに画像をインポートし，この PatternData オブジェクトに配列の要素を書き足すだけで実装できます．

このデータを用いて，ソースコード 4.1 に示したようなコンポーズ関数にも手を加え，柄変更機能を実装しました．柄変更機能は，当初の目標が達成された後，時間に余裕があったので追加で実装した機能です．構想だけは最初からあり，後から追加で実装したいと思いつつ主要な部分を実装していたため，追加で実装するのは比較的スムーズにできました．

```
1     object PatternData {
2         val patternData = arrayOf(
3             // No. 0
4             Pattern("default", null, R.drawable.ic_task_checked,
                  R.drawable.ic_task_unfinished),
5             // No. 1
6             Pattern("footprint", R.drawable.footprint_background,
                   R.drawable.footprint_checked, R.drawable.
                  footprint_unchecked),
7             // No. 2
8             Pattern("good", null, R.drawable.good_checked, R.
                  drawable.good_unchecked),
9             // No. 3
10            Pattern("sakura", R.drawable.sakura_background, R.
                  drawable.sakura_checked, R.drawable.
                  sakura_unchecked),
11            // No. 4
12            // Pattern( ... ),
13            // ... ここにデータを追加するだけで，柄を追加できる
14        )
15    }
```

ソースコード 4.4 柄のローカルデータであるオブジェクト PatternData

4.5 おわりに

夏，春と 2 回グループでアプリ開発をしてきましたが，春のアプリ開発では夏の経験を生かしてより短い期間でもアプリを完成させきることができ，良かったです．今回は，私は設計と実装のサポートなどに集中して，基本的に実装はグループメンバーにしてもらいました．最後，追加機能は自分も試しながら実装したところもありましたが，夏とは違うグループ開発の進め方に挑戦して結果も満足のいくものとなったので良かったです．

開発環境の構築から Android Studio の使い方およびアプリ開発の仕方まで，主に公式のチュートリアルを進めることでできることを増やしてきました．春のアプリ開発から，参考文献 [5] のようなページも参考にして開発に取り組むことができ，楽しいです．夏の時点では難しかった部分が，後期の授業で Java を使ったソフトウェアの開発を学んだこともあり少し分かるところも増えましたが，Lime の記事を執筆するのは難しかったです．使った Compose の説明や学んだことを中心に記事に書けなかったので，慣れてきたがまだ理解できていないところも多いなと改めて思いました．春のアプリ開発では，はじめは値保持機能なしでシンプルなアプリを考えましたが，なかなか良いアイデアは浮かびませんでした．ちょっとした機能が追加できるようになるだけで表現できるものがかなり増えるので，これからもできることを増やしていきたいです．

また，アプリ開発に限らず，勉強して部内に共有できるようにしたいと思います．最後まで読んでいただきありがとうございました．

参考文献

[1]「Compose を用いた Android アプリ開発の基礎」コース. Android developers https://developer.android.com/courses/android-basics-compose/course?hl=ja

[2] Jetpack Compose で 3 点ボタンのドロップダウンメニュー. Hatena Blog. 2022.2.4. https://koko206.hatenablog.com/entry/2022/02/04/024356

[3] androidx.compose.material. Android developer https://developer.android.com/reference/kotlin/androidx/compose/material/package-summary

[4] MaterialSymbols & Icons. Google Fonts. https://fonts.google.com/icons

[5] Buttons: floating action button. Material Design. https://m2.material.io/components/buttons-floating-action-button#usage

（いずれも最終閲覧日 2024 年 3 月 17 日）

5 久しぶりにphina.jsを触ってみた話

情報工学課程 1 回生 山下 京吾

5.1 はじめに

Lime をご覧の皆様、はじめまして。2023 年度入学生の山下と申します。この記事が世に出ている頃には 2 回生になっていることでしょう。

phina.js という JavaScript 製の国産ゲーム開発ライブラリがあります。（読みは「ふぃなどっとじぇーえす」）ゲームプログラミング初心者にも扱いやすいライブラリであり、またオープンソースなので誰でも開発に参加できるという特徴があります。中学・高校の頃、ホームページ [1] の「初心者でも手軽にゲーム開発ができる！」という旨の触れ込みに惹かれた私はこれを使ってゲーム制作の真似事をやったりしていたのですが、今回 Lime に記事を出すにあたって、久しぶりに phina.js を使って簡単なゲームを作ってみました。私個人としては数年ぶりの phina.js であり、今回のゲーム製作体験について、改めて phina.js を扱った所感とともに色々書いていこうと思います。技術的な話があまり無く恐縮ですが、読んでいただければ幸いです。

5.2 作ったゲームと制作の大まかな流れ

とりあえずゲームプレイのスクリーンショットを見てもらえればと思います。私が今回作ったゲームはこんな感じのゲームです。（図 1.1）ゲームは [2] から遊べるようにしています。（PC からのアクセスを推奨）

郵便局員の主人公が、紙を目当てにどんどんやってくる白ヤギをかわしながら 16 通の郵便を届ける..... という何のひねりもない内容のゲームとなっております。言い訳をすると良いアイデアが思い浮かびませんでした。ゲーム制作はアイデア出しが一番難しいです。制作に際しては、phina.js ひいては JavaScript を久しく扱っていなかったということもあり、ゲームとして最低限遊べる形にするのに割と時間がかかってしまいました。

制作の基本的な流れとしては、phina.js に実装されているクラスを用いてオブジェクトを作り、それをシーンと呼ばれるゲーム画面に表示される部分に追加していく、という感じで、特殊な実装はあまり必要ありません。

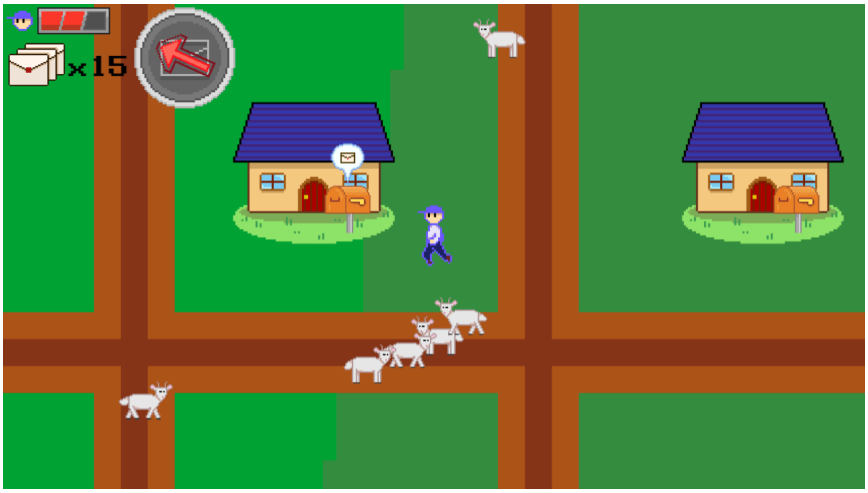

図 5.1　そこそこがんばった..... と思う

とはいえもう少し細かく書いてみましょう。phina.js では、既にあるクラスを継承して新しいクラスを作ることができます（シーンはクラス継承によって作られています）。例えばこのゲームでは Player クラスを作っており、実際には以下のように書いています。

```
1  phina.define('Player', {
2      superClass: 'RectangleShape', //四角形のクラス これが継承の対象
3      init: function() {
4          //ここでの this は Player クラス自身を指している
5          //superInit() で継承元クラスの初期化、引数を渡すこともできる
6          this.superInit({
7              width: 18, //横幅
8              height: 54, //縦幅
9              fill: 'transparent', //色 transparent... 透明
10             stroke: 0, //辺の太さ
11         });
12
13         //グラフィックを追加
14         this.image = phina.display.PixelSprite('mailman', 20, 30)
               .addChildTo(this).setPosition(this.x, this.y - 16).
               setScale(3, 3);
15         //グラフィックのアニメーションを設定
16         let animation = FrameAnimation('mailman_ss').attachTo(
               this.image);
17         this.animation = animation;
18         animation.gotoAndPlay('stand');
19
20         //プロパティ設定
21         this.health = 3;
22         this.isInvincible = false;
23         this.invincibleTime = 4000;
```

```
24          this.speed = PLAYER_SPEED;
25          this.rate = 1.5;
26          this.direction = 0;
27      },
28  });
```

ソースコード 5.1 main.js の一部

クラスを使ったオブジェクト生成はプロパティの設定・操作がしやすい他、書きやすさや可読性も良いです。新たに作ったクラスでオブジェクトを生成し、シーンに追加するにはこういう風に書きます。

```
1   phina.define('MainScene', { //メインとなるゲーム画面のシーン。この他にもシーンを複数作り、タイトル画面やリザルト画面を設定することもできる
2       superClass: 'DisplayScene', //シーンを作る場合、このクラスを継承する
3       init: function() {
4           this.superInit({
5               width: 1280,
6               height: 720,
7               backgroundColor: '#9ae28c', //背景色
8           });
9
10          let player = Player().addChildTo(this); //addChildTo しないと画面に表示されない
11          player.setPosition(640, 360); //シーンの中央に配置
12      }
13  });
```

ソースコード 5.2 example

ファイルの一番最後にはゲーム本体の設定を書きます。

```
1   //メイン処理
2   phina.main(function() {
3   //アプリケーション生成
4       var app = GameApp({
5           startLabel: 'title', //タイトルシーンから開始する
6           scenes: [
7               {
8                   label: 'title',
9                   className: 'TitleScene',
10                  nextLabel: 'main', //シーンを抜け出した時、どのシーンに向かうかを設定する
11              },
12              {
13                  label: 'main',
14                  className: 'MainScene',
15                  nextLabel: 'result',
16              },
```

```
17              {
18                  label: 'result',
19                  className: 'ResultScene',
20                  nextLabel: 'title',
21              }
22          ],
23          width: SCREEN_WIDTH, //画面の横幅
24          height: SCREEN_HEIGHT, //画面の縦幅
25          fps: 60, //FPSの設定
26          assets: ASSETS, //アセット（外部コンテンツ）に関する設定を記述したファイルの指定
27      });
28      //アプリケーション実行
29      app.run();
30  });
```

ソースコード 5.3 main.js の一部

ここまで書いたものを参考にして以下のように書いてみます。

```
1   // phina.jsをグローバル領域に展開
2   phina.globalize();
3
4   const SCREEN_WIDTH = 1280;
5   const SCREEN_HEIGHT = 720;
6
7   phina.define('MainScene', {
8       superClass: 'DisplayScene',
9       init: function() {
10          this.superInit({
11              width: SCREEN_WIDTH,
12              height: SCREEN_HEIGHT,
13              backgroundColor: '#9ae28c',
14          });
15
16          let player = Player().addChildTo(this);
17          player.setPosition(640, 360);
18      }
19  });
20
21  phina.define('Player', {
22      superClass: 'RectangleShape',
23      init: function() {
24          this.superInit({
25              width: 100,
26              height: 100,
27              fill: 'red',
28          });
```

```
29      },
30
31      update: function() { //毎フレーム実行したい処理をここに書く
32          this.rotation++; //rotation... オブジェクトの角度を表すプロパティ
33      }
34  });
35
36  phina.main(function() {
37      var app = GameApp({
38          startLabel: 'main',
39          scenes: [
40              {
41                  label: 'main',
42                  className: 'MainScene',
43                  nextLabel: 'main',
44              },
45          ],
46          width: SCREEN_WIDTH,
47          height: SCREEN_HEIGHT,
48          fps: 60,
49      });
50      app.run();
51  });
```

ソースコード 5.4　hoge.js

この hoge.js を埋め込んだ html を開くと、こんな風に表示されるかと思います。（図 1.2）

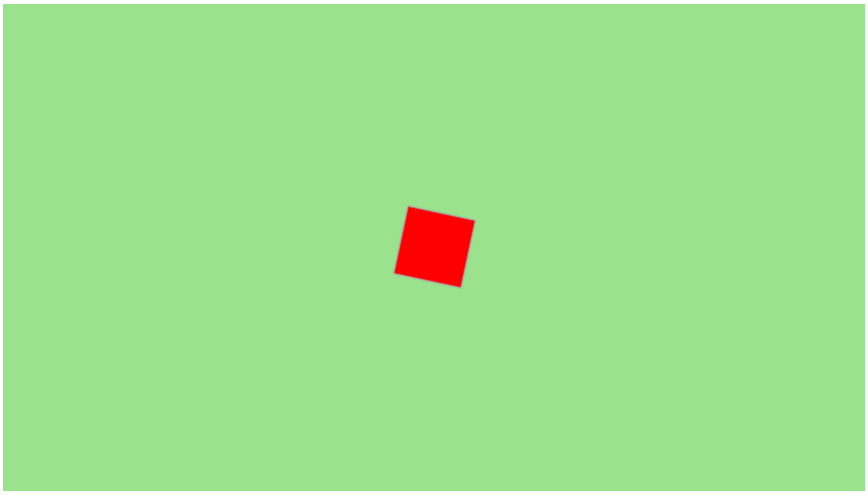

図 5.2　ゲーム画面、実際には図形がくるくる回っているはず

これをベースとして、ここにシーン・オブジェクト・アセット等を追加してあれこれすることでゲームを作っていきました。

（その他、制作に関する詳細な部分はゲームのソースコードに補足という形でコメントを残しているので、そちらを見ていただければと思います。）

5.3 苦労したところ

こんな単純なゲームを作るだけでも色んな所で苦労しましたが、最も大変だったのはスクロールの実装です。間違いなくこれに一番時間を掛けています。phina.js にはカメラ機能のようなものがない（はず）ので、欲しいなら自力で作ってやる必要があります。仕組みとしては、マップ全体に置かれているオブジェクトをまとめて一枚の画像にしてそれを移動させる、といったものを考えていました。幸いなことに、それに近いことをやっている記事 [3] があり、こちらが非常に参考になりました。

具体的には、[3] の「ちょっと楽なやり方」をベースに、毎フレーム scene._render() を実行するようにコードを加えたのみです。これ自体は至極単純な実装ですが、こうすれば良かったということに気付かず、私はこの形に落ち着くまでに結構な試行錯誤を繰り返していました。例えば「CanvasApp を使う方法」を試していた時期もあったのですが、そういう仕様なのか私の書き方が悪かったのか、addChildTo(scene) したオブジェクトで Tweener させようとすると、なぜか wait() が実行されないという事態になり、そこで結構悩み込んでしまったことがありました。Canvas 周りの機能を用いた実装は大抵ライブラリのソースファイル（とそこにコメントで書いてある簡易ドキュメント）とにらめっこしていた覚えがあります。

5.4 不十分な点

- 壁の衝突判定が未完成（図 1.3）
  単純に難しい！本来はちゃんと作りたかったのですが、四角形同士の判定ですら上手く組み上げることが出来ず、断念する形となりました。一応 Box2D を使えば衝突判定を持ってくることはできるものの、これを見下ろし型のマップに応用する方法が分からず..... 標準でそういった衝突判定が実装されているゲームエンジンは当たり前のようで実は凄いことなのだと感じました。
- 敵のレパートリーを増やしたい
  白ヤギさんがいるなら黒ヤギさんも、ということで作りたかったのですが、記事提出の締切の都合上、これを書いている時点では出来ていません。要は間に合わなかったということです。ちなみに白ヤギさんは郵便局員を普通に追いかけるだけなので、黒ヤギさんはロケットのように一直線に飛んでくる..... みたいな感じの動きをイメージしていました。

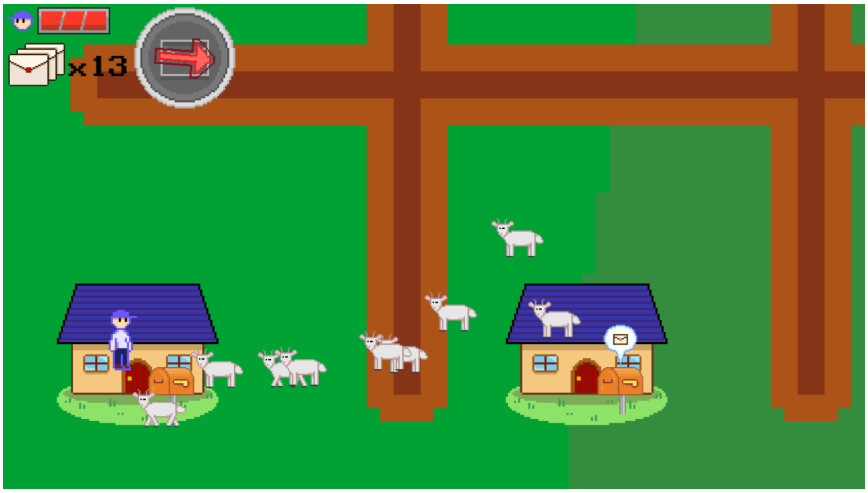

図 5.3　おうちにめり込む郵便局員と白ヤギさん

5.5　感想

ゲームを作るにあたって、有志の方々が書いてくださった沢山のチュートリアルや解説記事に助けられました。（例えば [4] など）こういった資料の多さも phina.js が初心者に優しい理由の一つなのだろうと思います。一方で、凝った機能や演出を作ろうと思うとやはり JavaScript（の主にクラス、Canvas に関する部分）に対する知識や、ソースファイル・ドキュメントを読む力がそれなりに必要にはなります。しかしそれは裏を返せば、初心者から上級者まで幅広く扱えるということです。そう考えると、phina.js は可能性に溢れた面白いライブラリなのだと改めて思います。

また、コーディングメインのゲーム制作はほぼゼロからゲームを組み立てている感覚があって面白いものです。1.3 項には書いていませんでしたが、その分デバッグはかなり大変で、この辺りはデバッグが円滑に進められる便利なシステムを自前で用意できれば良さそうだと感じました。

5.6　おわりに

ここまでお読みいただきありがとうございます。今回 phina.js に改めて触れてみようと思ったのは、もちろん Lime 記事のネタ目的でもあるのですが、同学の先輩が phina.js の記事を書いていた（その記事はインターネット上に公開されています）から、というのが実のところそもそもの理由でした。それまで自分の身の回りに phina.js を知っている人はいなかったため、その記事の存在に衝撃を受けるとともに、「これを機にもう一度やってみようかな」と思い立ったわけなのです。

ゲーム制作の一つの形として、phina.js の存在をより多くの人に知ってもらいたいと

思っています。前項でも述べたようにチュートリアルが充実しているため、簡単なゲームなら JavaScript の知識が無くとも作ることができると思います。phina.js を使ったゲーム制作は、phina.js の開発者が作った Web エディタの Runstant[5] でよく行われており、こちらを使えば実行環境の下準備等もほぼ必要ありません。

あなたも phina.js でお手軽にゲーム制作、やってみませんか？

参考文献

[1] https://phinajs.com

[2] https://recimode.github.io/game/1/
https://recimode.github.io/game/1/main.js - ゲーム本体のソースコード

[3] https://qiita.com/simiraaaa/items/7355ed8037ebd7d99bcb

[4] https://zenn.dev/alkn203/articles/phina-start-guide

[5] https://runstant.com

編集後記

編集を務めました石倉です。ここまで目を通していただき、ありがとうございます。

今年は記事の発行が例年より遅くなってしまいましたが、無事に Lime63 号を出すことができ良かったです。執筆のスケジュールが短い期間になってしまった中、執筆者の皆さんは記事を書いていただき、ありがとうございます。編集作業は初めてだったのですが、先輩方が編集作業用のデータをまとめてくださっていたおかげで、スムーズに進めることができました。感謝申し上げます。

令和 6 年 3 月 24 日
編集担当　石倉 朋佳

Lime Vol. 63
令和 6 年 3 月 24 日発行 第 1 刷

発行 京都工芸繊維大学コンピュータ部
Web サイト: http://www.kitcc.org/
電子メール: question@kitcc.org