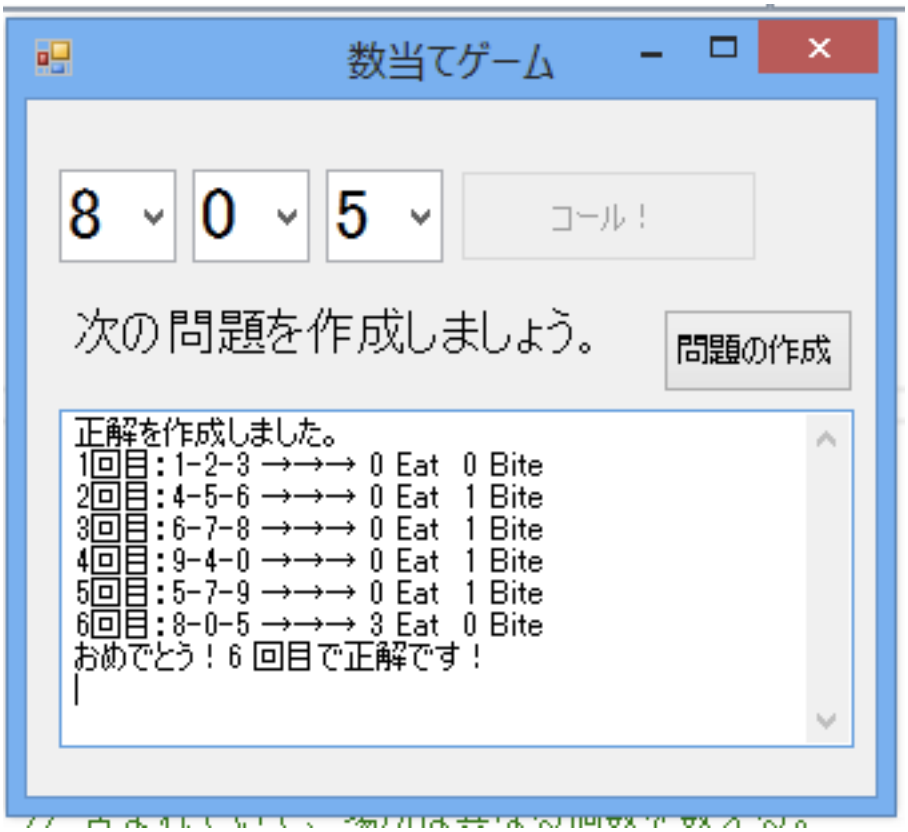
反復と配列、ついでに効果音再生

今回は、数当てゲームを作ります。



今日の課題は、上のようになります。「3 Eat」で効果音（ファンファーレ）がなるようにプログラムします。（効果音再生に必要な記述は、たったの４行です。）

最初に、どんなゲームか説明します。

相手（コンピュータ）が、３つの異なる数字の列を「正解」として合成します。それを当てるだけのゲームですが、毎回、「これが正解かな」という数字を相手に伝えます。

判断する時は、数字の位置も関係します。同じ数字が同じ位置にあったなら、「Eat（イート）」と言います。Eat（イート）が一つなら、１Eat（ワンイート）、二つなら２Eat（ツーイート）、三つなら３Eat（スリーイート）です。３Eatは、「完全一致」ということですから、正解です。

位置は違うけれども、同じ数字がどこかにあったならば、「bite（バイト）」と言います。上記の例では、「０」の数字が異なる位置にあり、それが一つだけなので、１bite（ワンバイト）と言います。（複数形のｓはつけていません!）

正解するまでの回数を競う、というよりも、先攻後攻を決めて、先に 3 Eat した方が勝ち、というゲームです。（どこかでご存知かも・・・）

今回、新しく学ぶのは「Combo Box」（コンボボックス）と配列、そして、for-loop と、while-loop です。

ComboBox は、画面上の部品として使ってみてください。使いやすいと思えば使えばいいし、もっと他のやり方があると思ったら、それでも構いません。
TextBox による手入力でも、結果は同じです。ただ、Combo を使うと、タッチパネルなどでも扱いやすくなります。

また、効果音再生は、たったの 4 行でプログラムが面白くできますから、そのために教材に含めました。Total の行数は約 140 行です。
結構長いので、プログラムの部分部分は、別に掲示します。

大切なのは、配列と、for-loop, while-loop です。if や switch/case などと並んで、プログラムを構成するのになくてはならない部品です。使いこなせるように理解してください。

ただ、毎回同じことを言いますが、たった一度の授業で完全にマスターする必要など何もありません。何回も、何十回も使っているうちに見慣れて、自然に書けるようになれば、それでプログラマとしては「入門者」を卒業して「初級プログラマ」になれます。

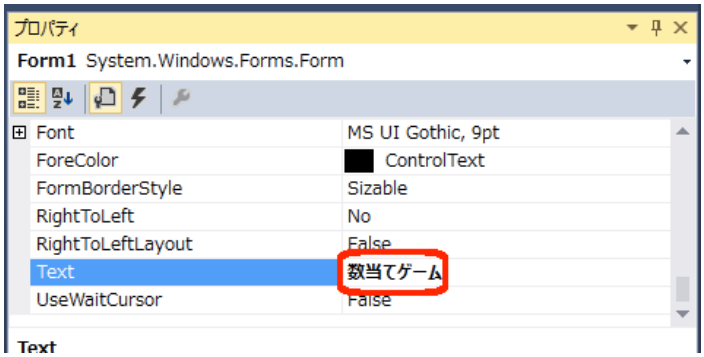
また、プログラマにならなくても、プログラムを読めばわかるし、それなりに意味がとれるようになれば、臨床工学技士の仕事についても、きっとプログラミングのスキルが役に立つと思います。
まず、見慣れて、見てわかることが大切です。

ただ、「自分には出来る」と思って、実際に資料を読んで、手を動かして下さい。せっかく書いてあるのに、読まずに作業している人があまりにも多くて・・・（涙）。読まなかったらできないでしょう!?

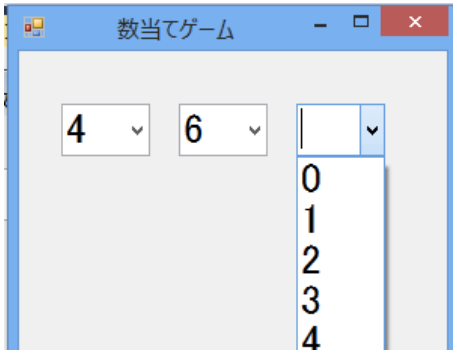
プロジェクトの生成と画面の準備

既に何度か説明していますので、プロジェクトの生成については省略させてください。「新規プロジェクトの生成」を行います。私は、「kazuate」プロジェクトと名付けました。
ここで、画面（Form）の設計を行います。

最初に、Form の見出しを設定します。
私は、「数当てゲーム」としました。
どんな名前をつけても構いませんが、そのプログラムが実行しているとき、少なくとも自分には意味がわかる名前にして下さい。

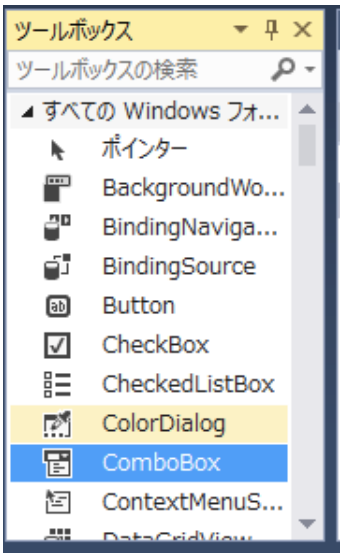


見出しを設定するのは Text です。
Name ではありません。
Name の方を変えてしまうと、エラーが出て結構大変だと思います。Name をつけかえてしまってエラーが出た人は、すぐに元に戻し、消せないエラーがあったらすぐに挙手してください。



今回は、ComboBox の導入があります。
ComboBox とはどんな画面かと言えば、「下向き矢印（▼）」をクリックすると、選択可能な値が選べる、そんな画面部品です。
右に、今回使用する ComboBox の実行例を記しました。
予め、0～9 までの数字を登録しておきますが、数字を「入力」する代わりに、選択して入力させます。
おそらく皆さんも、「北海道、青森県、岩手県、秋田県・・・沖縄県」などと、都道府県を入力するのに ComboBox が使われていたり、あるいは、音楽チケットをとる際に「S 席、A 席、B 席、C 席」などと、予め選択するものを ComboBox から選択したりした経験があると思います。
今回は、これを使います。

この ComboBox も、Button や Label などと同様に、ToolBox から選択することができます。
毎回同じ説明を行っていますが、画面デザインは「こうでなければならない」ということは全くありません。
自分の好きな場所、全体のデザインをイメージして、自分の好きなようにアレンジして、配置してください。



皆さんが「中級」になってきたら、こうした「デザインツール」を使うのではなく、座標計算をして、プログラムによって画面上の特定の場所に貼付けていくことができる部品です。ですが、慣れるまでは、デザインツールを使って、画面に部品を「置いて」いって下さい。

ここで、前回と同様に、画面の部品に名前をつけていきます。
以下は、私の画面の部品の説明です。

型	Name	内容
ComboBox	dig3	３桁目の数字(dig は Digit ; 桁)
ComboBox	dig2	２桁目の数字
ComboBox	dig1	１桁目の数字
Button	callButton	コール（カードゲームで「これでどうだ」的な声）をするボタン
Button	startButton	コンピュータの問題を作らせて、ゲームを開始するためのボタン
Label	messageLabel	「数字を選んでください。」と 「数値が設定されていません」の表示用。
TextBox	history	それまでのコールごとの結果を記録するメッセージ表示用の Box.

部品の名前は自分で変えて構いませんが、C 言語の予約語の言葉は使えません。
皆さんは、自分の画面のデザインとプログラムに合わせて、こうした表を、自分で作成して下さい。
皆さんがもしもプロフェッショナルだったなら、たぶん、「開発ノート」などにこうした「一覧表」を作成します。（こうした表を用意しておかないと、スムーズにプログラムが書けません。）

改めて復習です。名前に使えない「C 言語の予約語」です。

C 言語の予約語

以下の用語は、C 言語の中で使うと特別な意味や役割を持ってしまいます。
名前に使えません。今後、必要な言葉は段階的に授業で扱います。

```
auto    break   case    char    const   continue        default
do      double  else    enum    extern  float   for     goto
if      int     long    register        return  short   signed
sizeof  static  struct  switch  typedef         union   unsigned
void    volatile        while
```

これらの言葉以外を名前に使って下さい。

今回の配置は、以下のような感じにしてみました。

部品にきちんと名前をつけない人が多すぎます。

名前を奪われたプログラムを書いた人は、~~人間界に戻れません~~単位がもらえません。

音源について

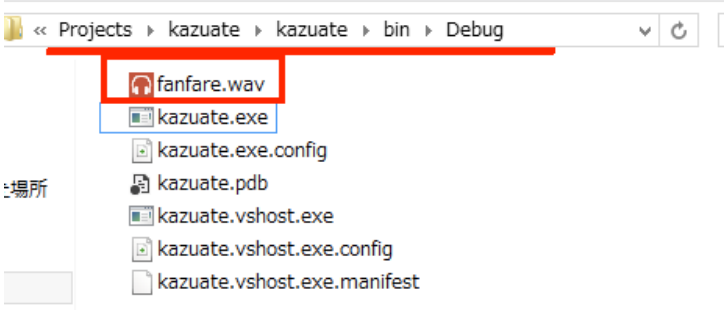
今日の教材では、「正解音」としてファンファーレを流します。
C# 環境の、SoundPlayer クラスを使います。このクラスでは、MP3 フォーマットの音は再生出来ません。
WAVE フォーマット（拡張子が.wav）の音ファイルだけが再生可能です。

転載可能な音源を探しましたが、今年の教材は音素材としてみゅーさんのサイトから音源を頂きました。この方です。厚く御礼申し上げます。
http://www.ne.jp/asahi/music/myuu/ (2014 年 5 月 17 日参照)
http://www.ne.jp/asahi/music/myuu/profile/profile.htm
WAVE ファイルは、以下のリンクからダウンロードしました。
http://www.ne.jp/asahi/music/myuu/wave/wave.htm

学生の皆さんが各自ダウンロードして使う場合には他にも使えるサイトが多数ありますので、各自検索してみて下さい。（私の指導書やサイトに転載はできないものが多いため、紹介もしませんがご了承下さい。）MP3 から WAV への変換は、各自で検索して調べて下さい。

ただ、あまりに説明を聞かない人が多いので「MP3 のファイルをダウンロードして組み込んだが、再生出来ない。どうしたらいいか」という質問には一切対応しませんので、覚悟して作業して下さい。
（この説明をしているのに、MP3 が再生出来ないというトラブルで質問する学生が、たぶん 4 人くらいはいるのではないかと・・・）

また、ダウンロードしたファイルは、プロジェクトの bin￥Debug のディレクトリにコピーして下さい。これ以外の場所では、実行時にエラーになります。



今回のプログラムでは、もう一つ一覧表にして整理してみます。変数の一覧です。何の目的で、どんな風に「変数」を使うか、必ず一覧にして、言葉にしてまとめて下さい。この習慣を身につけてください。プログラムの開発環境に、こうした一覧表の整理機能がある場合は、それを使います。ない場合でも、簡単なコメントをプログラムに書き込んでください。

型	Name	内容
int[] (サイズ 3)	ans	コンピュータが用意した正解
int[] (サイズ 3)	trial	ユーザが入力した数
int	eat	場所まで一致した Eat の数
int	bite	数字が含まれている Bite の数
int	turns	「何回目の挑戦か」の回数
Random	rnd	乱数を管理する。
SoundPlayer	sp	音楽再生用の部品 3-Eat で当てた時に流れる音楽を保持する

今回使う変数は、7 個だけですが、それぞれの役割や、「どんな情報を持っているのか？」を考えながら使ってください。

```
int[] ans = new int[3];
int[] trial = new int[3];
int eat, bite;
int turns;
System.Media.SoundPlayer sp;
Random rnd;
```

入力後の画面は、こんな感じになります。

```
11  namespace kazuate
12  {
13
14      public partial class Form1 : Form
15      {
16          int[] ans = new int[3];
17          int[] trial = new int[3];
18          int eat, bite;
19          int turns;
20          System.Media.SoundPlayer sp;
21          Random rnd;
22
23          public Form1()
24          {
25              InitializeComponent();
```

まず、

int[] ans = new int[3];

という宣言文で、ans の名前を宣言します。今回使う ans や trial は「配列」と呼ばれるプログラム要素です。

今回のプログラムでは、デストラクタを使う要素はありません。

今日は「配列」というのがどんなものか、その雰囲気に目を慣らしてください。

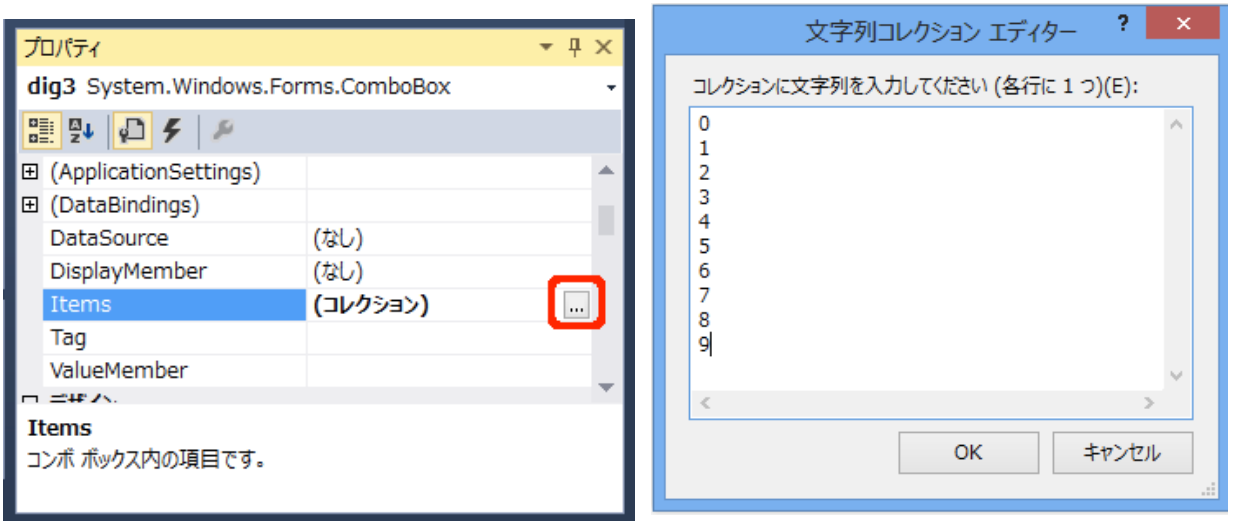
Form1 では、今回も使用する「乱数」の初期化と、SoundPlayer の初期化を行っています。

予め「音」ファイルを指定して読み込んでおいて、再生するときは Play()メソッドをコールするだけで使います。

```
public Form1()
{
    InitializeComponent();
    rnd = new Random(Environment.TickCount);
    sp = new System.Media.SoundPlayer();
    sp.SoundLocation = "fanfare.wav";
    sp.Load();
}
```

画面を仕上げてしまいましょう。

最初に貼付けた ComboBox について、プロパティを見てみます。



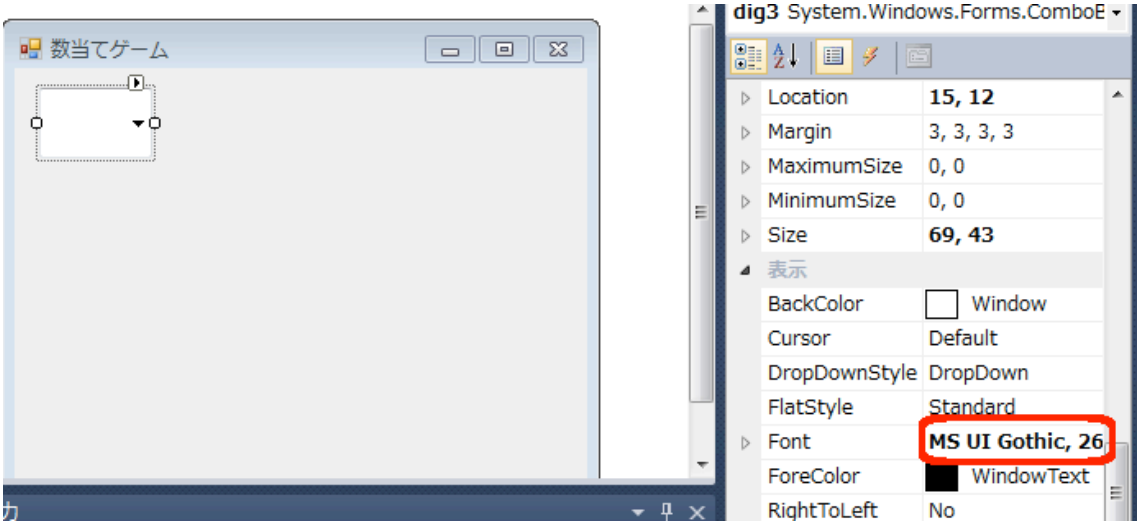
Items（画面に表示する選択肢を設定する部分）で、右側の[…]ボタンを押します。

そうすると、「文字列コレクションエディタ」が表示されます。

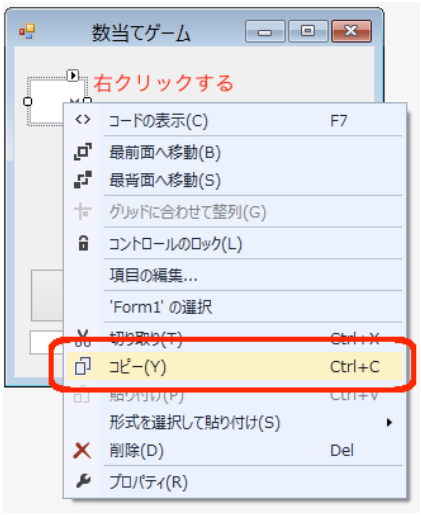
ここで、０から９までの数字を入力します。今回も画面から入力した値を「数値」として取り込んで処理するので、半角の文字で入力します。

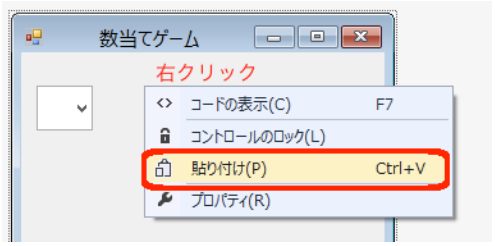
入力したら、[OK]ボタンを押してください。

私は文字の大きさを変えました。



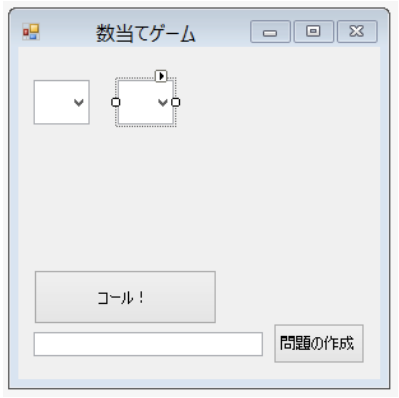
最初の一つを入力したら、フォーム[デザイン] の画面で設定が終わった ComboBox をコピーし、フォームの中で貼付けします。

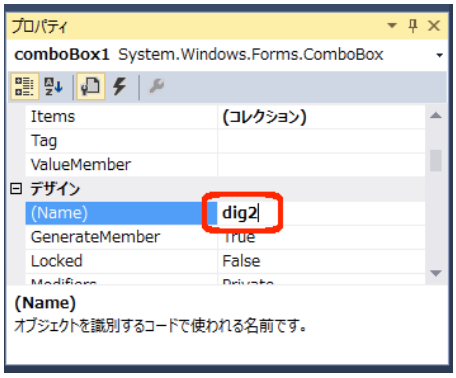


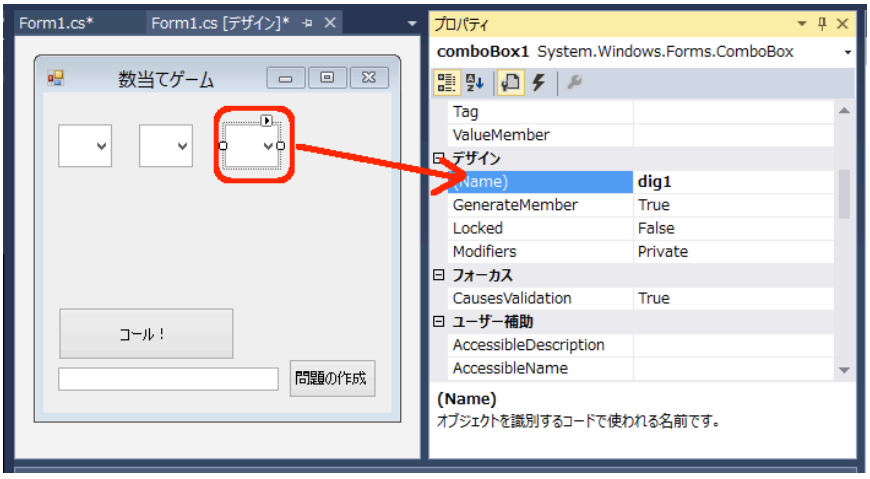


そうすると、最初の一つに行った設定内容が、そのまま貼付けされた ComboBox にも引き継がれます。

そうしておいて、名称を dig2 などのように、最初に画面設計した時の名称につけ替えます。

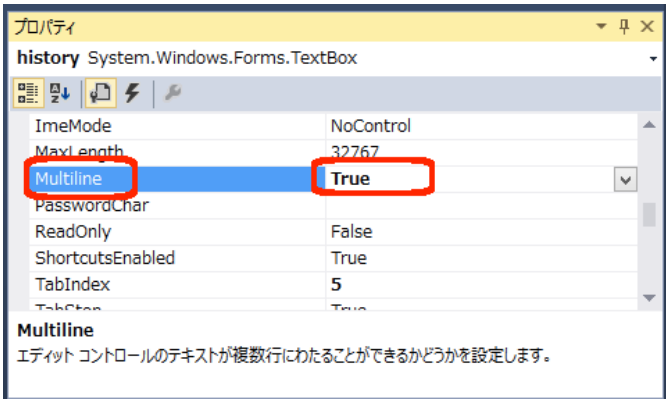






この後、コールボタン（callButton）や、スタートボタン（startButton）、メッセージラベル（messageLabel）などを画面に配置し、名前と Text を設定していって下さい。（この作業は、前回までのプログラムを参照して下さい。）

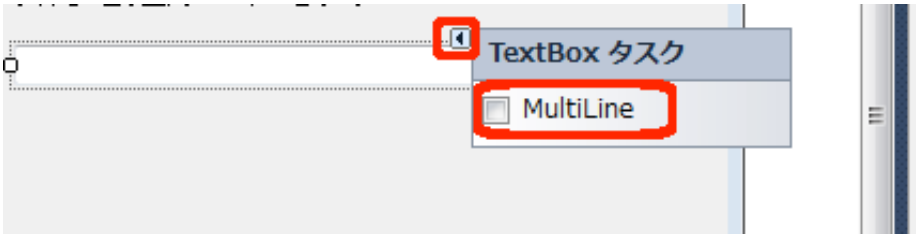
左下の位置にテキストボックスを配置し、history と名前をつけます。
数当てゲームでは、これまでどんな「数」を伝えて、その結果がどうだったかの記録が大切です。これを一目でみられるようにしますが、１行だけのテキストボックスだと一覧できません。そこで、マルチライン（複数行表示）にします。



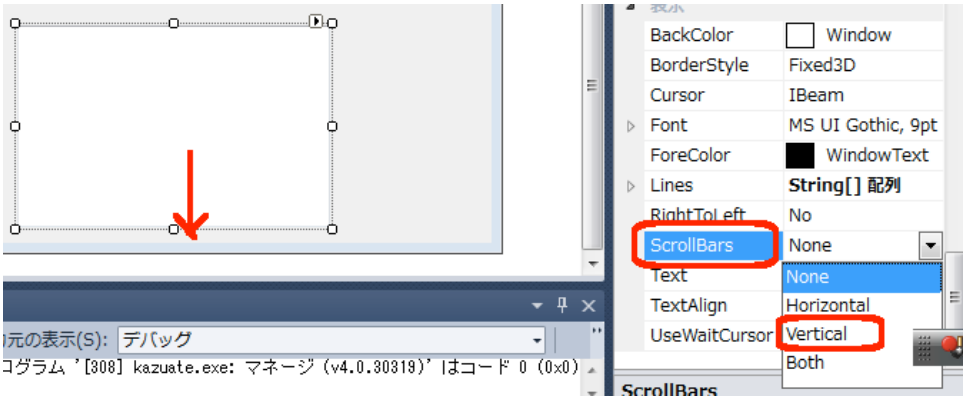
前回とはちょっと最初の画面デザインのやり方や、テキスト内容の表示方法が変わっていますので、注意してください。

history は、プロパティ「動作」から「Multiline」を「True」に設定します。
そろそろ、独自のデザインを進める人も出て来ていると思います。基本形は提示しますが、各自が自由に、これらと同等の画面要素を、レイアウトしていってください。

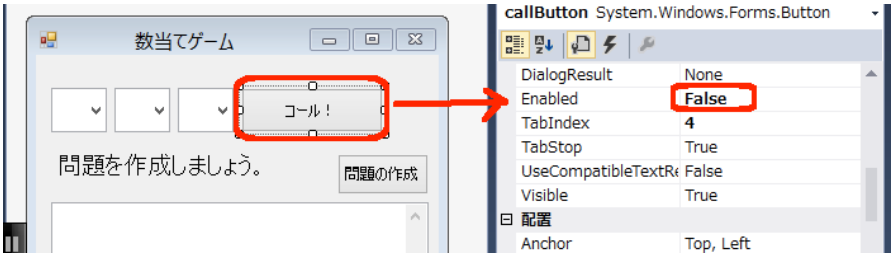
また、テキストボックスの右上に、小さいマークがあります。これをクリックしても、マルチライン表示の選択ができます。



マルチラインにすると、縦方向にサイズをかえられるので、縦長の入力 Box にします。また、縦方向のスクロール Bar を有効にします。
（Horizontal は水平、Vertical は垂直です。）



なお、コールボタンは最初は Enabled を false にしておきます。
問題を作成するまでは「コール」できないからです。



全部のボタンの配置が終わると、下図の感じになります。

ここで、「開始」ボタンをダブルクリックして、コーディング（プログラムを書くこと）を始めましょう。

コンピュータ側の正解を作ろう

開始ボタンをクリックした時の処理は、以下のようにプログラムを書きます。

```
turns = 0;
eat = 0; bite = 0;
history.Text = "";
setMessage(makeAns());
messageLabel.Text = "数字を選んでください。";
```

前後を合わせて画面を添付すると、下記のようになります。

```
70
71      private void startButton_Click(object sender, EventArgs e)
72      {
73          turns = 0;
74          eat = 0; bite = 0;
75          history.Text = "";
76          setMessage(makeAns());
77          messageLabel.Text = "数字を選んでください。";
78      }
79
```

turns は、何回目の挑戦かを記録しますが、これを 0 にリセットします。

また、履歴表示（history）の Text に、””（空白文字列）を代入して、表示をクリアします。

その後で、makeAns()をコールし、その結果を setMessage()に手渡しますが、次にこれらの関数を見てみましょう。

ここで呼び出す関数を、整理してみましょう。

関数名	setMessage( )
手渡す値	System.String　str
戻って来る値	なし
処理する内容	Str の内容を、history->Text に連結し、さらに改行をつなげる。（history の表示に str が追加される。）
使っている機能	特になし

```
History.Text = history.Text + str + "¥r¥n";
```

¥r¥n は改行を意味します。第 2 回の授業資料を見て下さい。

```
79
80      private void setMessage(String str)
81      {
82          history.Text = history.Text + str + "¥r¥n";
83      }
84
```

次に、正解を作る関数の作成を行います。

関数名	makeAns（）
手渡す値	なし
戻って来る値	System.String str 問題を作った、という画面表示用の文字列。 開発中には、実際に作成した答えを表示してしまう。 最後に、答えを表示しないようにする。
処理する内容	互いに異なる３つの数字を作成して、 int[] の ans[0], ans[1], ans[2]に代入する。
使っている機能	Random.Next()　乱数を発生する。 →　毎回異なる値となるようにする。

乱数の初期化（シードを与える）のは、以下の１行です。`Form1` コンストラクタに書かれています。

```
rnd = new Random(Environment.TickCount);
```

プログラムが起動された時刻（`1/100` 秒刻み）で乱数を初期化するので、毎回異なる値で始まります。この１行を

```
rnd = new Random(0);
```

にするとどうなるか、調べてみて下さい。毎回同じ正解になります。

プログラムの全体は、以下のようになります。

```
85      String makeAns()
86      {
87          int     i, j, num;
88          bool    success = false;
89
90          while( !success ){
91              for( i=0; i<3; i++ )
92              {
93                  num = rnd.Next(0,10);
94                  ans[i] = num;
95              }
96              success = true;
97              for( i=0; i<2; i++ )
98              {
99                  for( j=i+1; j<3; j++ )
100                 {
101                     if( ans[i]==ans[j] )
102                     {
103                         success = false;
104                         break;
105                     }
106
107                 }
108             }
109         }
110         callButton.Enabled = true;
111         // return  String.Format("正解は{0}-{1}-{2}です。", ans[2], ans[1], ans[0] );
112         return "正解を作成しました。";
113     }
114
```

さてここで、本日の真打ち（for-loop と while-loop）が登場しました。

```
// コンピュータ側の正解を作る
        String makeAns()
        {
            int     i, j, num;
            bool    success = false;

            while( !success ){
                for( i=0; i<3; i++ )
                {
                    num = rnd.Next(0,10);
                    ans[i] = num;
                }
                success = true;
                for( i=0; i<2; i++ )
                {
                    for( j=i+1; j<3; j++ )
                    {
                        if( ans[i]==ans[j] )
                        {
                            success = false;
                            break;
                        }

                    }
                }
            }
            callButton.Enabled = true;
            return  String.Format("正解は{0}-{1}-{2}です。",
                                  ans[2], ans[1], ans[0] );
            // return "正解を作成しました。";
        }
```

順に見ていきましょう。

```
        for( i=0; i<3; i++ ){
            num = rnd.Next(0,10);
            ans[i] = num;
        }
```

この部分では、整数変数 **i** の値を **0** から **2** まで、１ずつ増加させます。
それぞれに、**0** 以上 **10** 未満の乱数を生成して、代入します。計算した結果は、**ans[i]**に格納されています。

for(式１ ; 式２ ; 式３) { ブロック }

では、ブロックの内容が反復されます。
式１は、ブロックの中を実行する前に行われる処理。
式２は、ブロックの中を継続実行するための条件で、ブロックの中を実行する前に判定されます。
式３は、ブロックの中を実行した後に行われる処理
です。
最初に i=0 が実行されます。反復条件は i<3 なのですが、繰り返し i++で i は１ずつ増加しますので、i が 0 から始まって、1 では実行、２でも実行されますが、３になると条件を満たさなくなって、この for-loop を抜けます。

配列の大きさが３だ、ということに気をつけてください。配列の大きさ（３）と同じ値をループ反復の条件のところに不等号で持って来ると、配列全部の要素に対して loop のブロック処理が実行できることがわかります。

```
num = rand() % 10;
```

の行では、「乱数」の値（rand()）を１０で割ってあまりを求めます。
10 で割った余は、0～9 の値となります。
生成した値を、array ans[i]に代入します。

ところが、このループの外側に

```
while( !success ) {
}
```

という条件がかぶさっています。これは何かと言えば、「同じ数字が使われていない」ということの確認のためです。

while(条件) { ブロック }

条件を満たしている間は、ブロック内を反復する。

最初の１回目では、「まだ正解が作成されていない」ので success に false を代入しておきます。

この後、for-loop で確認が行われます。

```
success = true;
for( i=0; i<2; i++ ){
    for( j=i+1; j<3; j++ ){
        if( ans[i] == ans[j] ){
            success = false;
            break;
        }
    }
}
```

外側（最初）の for-loop で、i を 0～1 まで変化させます。Ans[i]を相手と比較します。その比べる相手は、内側の for-loop で j を i+1 から 2 まで変化させます。

この i と j の変化する範囲を、実際にたどってみて下さい。比較する組み合わせは、

```
{ ( i, j ); ( 0, 1 ), ( 0, 2 ), ( 1, 2 ) }
```

これ以外の組み合わせは、ありません。

もし、ans[i]と ans[j]が等しかったら、 success = false; で、success に「失敗」を代入します。そして、もう一度、while(!success)のループを先頭から実行します。作成した正解の数字が全部違っていたら、success = true のままになって、while ループを終了します。

いくつも、新しい記述が登場しました。

まず、

```
if ( ans[i] == ans[j] )
```

で登場した、等号二つ(==)です。これは、比較演算子といって、左右が等しい場合に true となる論理演算子です。等号が一つだけの場合には、代入になってしまいます。(初心者がエラーを出しやすいところなので、注意してください。)

比較演算子

C 言語の比較演算子

名称	構文
小なり	a < b
小なりイコール	a <= b
大なり	a > b
大なりイコール	a >= b
非等価	a != b
等価	a == b

while(!success) の !(感嘆符；Exclamation mark; びっくりマーク)は、NOT(否定)を意味する否定演算子で、単項演算子です。
「success の NOT(否定)が、true だったら、while ブロックを反復する」
NOT（否定）が true ということは、false ですから、読み替えると
「success が false だったら、while ブロックを反復する」 となります。

break;文が登場しました。先週の登場場面では、break; によって switch()ブロックの中から離脱することを意味していましたが、今回の場合には、while ブロック、または、for ブロックのどちらかいずれか、最も break 文に近いブロックを抜け出すことになります。
意味は、「一つでも、数字の一致があったなら、もう失敗ははっきりしているから、内側の for ループを抜け出す」ということになります。

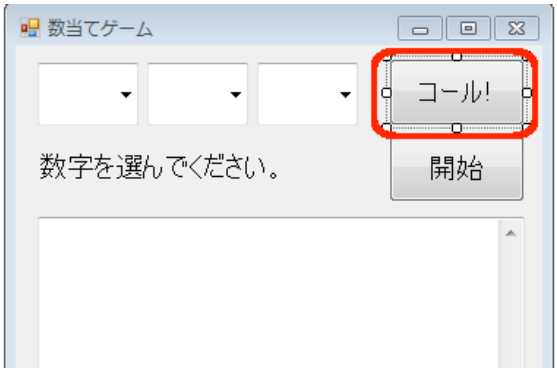
もしも、一つも数字が一致していなかったなら、問題作成に成功していますので、この確認の for-loop の直前で代入された success = true が有効となって、そのまま while ブロックの一番したまで実行され、その時は
while(!success)の条件が、false(true の否定)となりますので、それで「完了」します。

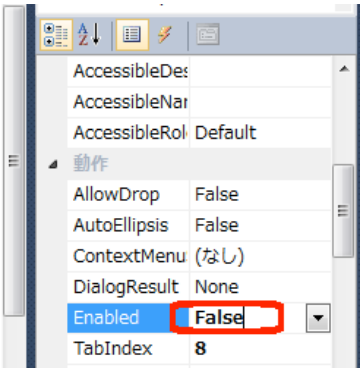
問題作成成功後の処理

```
    callButton->Enabled = true;
//    return  "正解を作成しました。"
    return  String::Format("正解は{0}-{1}-{2}です。", ans[2], ans[1],
ans[0] );
}
```

一つは、「コールボタン」を有効にしています。

実は、問題が作成されないうちにコールボタンが押されるとエラーになってしまうため、コールボタンは「利用不可」（Enabled = false）にしておきます。





それが、「問題作成の成功」で　Enabled = true と代入して、ボタンが使えるようにしたのです。

関数の呼び出し元に返す文字列は、「本番」では「正解を作成しました。」ですが、開発中は、正解が丸見えになるようにしておき、//　（注釈行）の切り替えで、戻せるようにしてあります。

コール（正解確認）ボタンの処理

次は、コールボタンの処理です。入力後の画面は、以下のようになります。

```
33          private void callButton_Click(object sender, EventArgs e)
34          {
35              bool   result;
36              Int32  value;
37
38              result = Int32.TryParse( dig1.Text, out value );
39              if( result )
40              {
41                  trial[0] = value;
42                  result = Int32.TryParse( dig2.Text, out value );
43              }
44              if( result )
45              {
46                  trial[1] = value;
47                  result = Int32.TryParse( dig3.Text, out value );
48              }
49              if( result )
50              {
51                  trial[2] = value;
52                  setMessage( checkAns() );
53                  if (eat == 3)
54                  {
55                      setMessage(String.Format("おめでとう！{0} 回目で正解です！", turns));
56                      sp.Play();
57                      callButton.Enabled = false;
58                      messageLabel.Text = "次の問題を作成しましょう。";
59                  }
60                  else
61                  {
62                      messageLabel.Text = "数字を選んでください。";
63                  }
64              }
65              else
66              {
67                   messageLabel.Text = "数値が設定されていません。";
68              }
69          }
70
```

ソースコードは、以下の通りです。

```
        private void callButton_Click(object sender, EventArgs e)
        {
            bool   result;
            Int32  value;

            result = Int32.TryParse( dig1.Text, out value );
            if( result )
            {
                trial[0] = value;
                result = Int32.TryParse( dig2.Text, out value );
            }
            if( result )
```

```
            {
                trial[1] = value;
                result = Int32.TryParse( dig3.Text, out value );
            }
            if( result )
            {
                trial[2] = value;
                setMessage( checkAns() );
                if (eat == 3)
                {
                    setMessage(String.Format("おめでとう！{0} 回目で正
解です！", turns));
                    sp.Play();
                    callButton.Enabled = false;
                    messageLabel.Text = "次の問題を作成しましょう。";
                }
                else
                {
                    messageLabel.Text = "数字を選んでください。";
                }
            }
            else
            {
                 messageLabel.Text = "数値が設定されていません。";
            }
        }
```

if や TryParseは既に電卓の時に説明しましたので、ここでは説明は省略します。途中で、resultがfalseになると（どこか数字が設定されていないと）、そこから後はTryParseは実行されず、最後のelseに来て、「数値が設定されていません。」というメッセージが表示されます。

dig1 の ComboBox の値は trial[0], dig2 の comboBox の値は trial[1]に、

dig3 の ComboBox の値は trial[2]にそれぞれ記録されます。
trial[0]～[2]に値が格納されている状態で、checkAns()を呼び出します。

checkAns では、コンピュータが用意した「正解」とユーザが入力した数値とを比較して、Eat と Bite を求めます。。

関数名	checkAns（）
手渡す値	なし
戻って来る値	System.String　str 結果の表示用文字列 “３回目：3-5-1　→→→　１Eat１Byte”など
処理する内容	コンピュータの用意した答えと、ユーザが入力した答えを比較して、Eat と Bite　を出す。
使っている機能	特になし

ソースコードは以下の通りです。

```
// 答えと照合する
        private String checkAns(){
            int i, j;
            eat = 0;

            // 場所も数値も一致している個数を数える。
            for( i=0; i<3; i++ )
            {
                if( ans[i]==trial[i] )  eat++;
            }
            // 含まれていて、場所は異なる個数を数える。
            bite = 0;
            for( i=0; i<3; i++ )
            {
                for( j=0; j<3; j++ )
                {
                    if( i==j )  continue;   // ループは反復するが今回は処理しない。
                    if( ans[i] == trial[j] )    bite++;
                }
            }
            turns++;
            return  String.Format( "{0}回目：{1}-{2}-{3} →→→ {4} Eat  {5} Bite",
                        turns, trial[2], trial[1], trial[0], eat, bite );
        }
```

入力後の画面は、以下のようになります。

```
115         private String checkAns(){
116             int i, j;
117             eat = 0;
118 
119             // 場所も数値も一致している個数を数える。
120             for( i=0; i<3; i++ )
121             {
122                 if( ans[i]==trial[i] )  eat++;
123             }
124             // 含まれていて、場所は異なる個数を数える。
125             bite = 0;
126             for( i=0; i<3; i++ )
127             {
128                 for( j=0; j<3; j++ )
129                 {
130                     if( i==j )  continue;   // ループは反復するが今回は処理しない。
131                     if( ans[i] == trial[j] )    bite++;
132                 }
133             }
134             turns++;
135             return  String.Format( "{0}回目：{1}-{2}-{3} →→→ {4} Eat  {5} Bite",
136                         turns, trial[2], trial[1], trial[0], eat, bite );
137         }
```

この、最初のループは **eat** を数えます。

```
eat = 0;
for( i=0; i<3; i++ ){
    if( ans[i]==trial[i] )  eat++;
}
```

予め **eat** を 0 に設定しておいて、

i	0	1	2
ans			
Trial			

Ans[0]と **trial[0], ans[1]**と **trial[1]**などの数値が等しいかどうかを確認し、等しかったら **eat** を 1 加算します（インクリメント）。

この **for-loop** では、**i** を最初 **0** に設定し、比較が終わったら 1 ずつ加算しながら **3** より小さい間（つまり **2** まで）変化させて比較を行います。

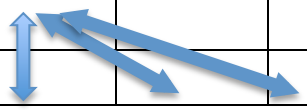
次の、bite のループは２重になっています。

```
bite = 0;
for( i=0; i<3; i++ ){
    for( j=0; j<3; j++ ){
        if( i==j )   continue;
        if( ans[i] == trial[j] )      bite++;
    }
}
```

まず、i を 0 から２まで（３より小さい間）、１ずつ変化させます。

予め eat を０に設定しておいて、

i	0		
j	0	1	2
ans			
Trial			

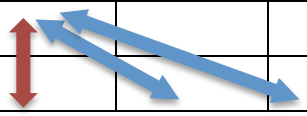


ループの１回目では、ans[0]と、trial[0], trial[1], trial[2]の比較を行います。ですが、

if(i==j) continue;

という文があります。これは、同じ位置だった場合には、何もしないで、最も内側のループで（つまり、j を変化させるループで、）その次をスキップして次に移れ、という命令です。

i	0		
j	0	1	2
ans			
Trial			



つまり、i が 0 で、j も 0 で一致している部分は、（Eat にカウントされているので、）Bite にはカウントせずに、「何もしないで、ループの次（j=1）に移れ」という命令になります。

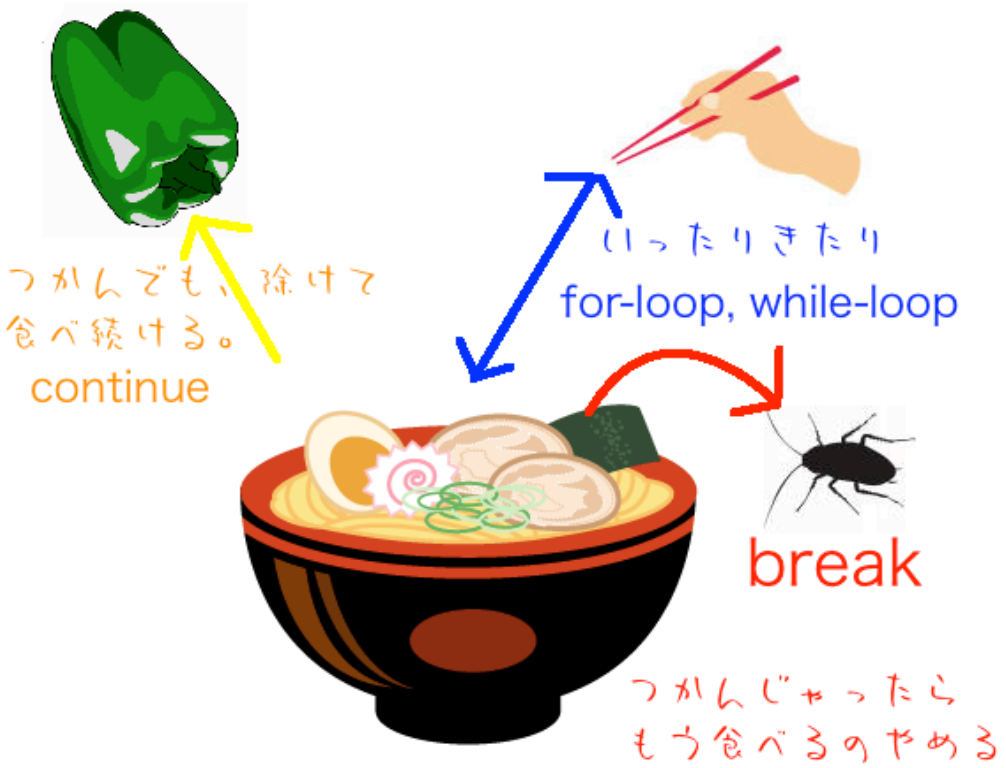
Break と continue の違い、ですが・・・

例えば、「料理を食べていて、箸を１回、１回、皿でつかんでから口に運ぶ」というのがループ（反復）でプログラムされているとします。

そこで、あなたの箸が、「ピーマン」をつかんだとします。あなたは、ピーマンが大嫌いです。だから、ピーマンは口に運ばずに、そこで「口に運ぶ」のをやめて、次の何かを箸でつかむ、ということを行います。これが continue です。

「とりあえず、この【ループの要素】はここで処理を打ち切るけれど、その次は、気を取り直して、続けます・・・」というのが、continue です。

ところが、あなたの箸が「ゴキブリをつかんだ」とします。こんなもん、これ以上食えるか！となったとします。そうなると、「箸を１回、１回運ぶ」という動作をそれ以上は完全にやめて、「もう。何も食わない」となってしまいます。これが　break; です。Break したら、もう何も続けません。



さて、ここでは、i=0, j=0　で continue しました。

ということは、内側のループを抜けるから、j を 0 で調べるのは、もうこれで打ち切ります。ピーマンはやめましたが、次の j=1 から 2 まで変化させる部分はこのまま続けていきます。

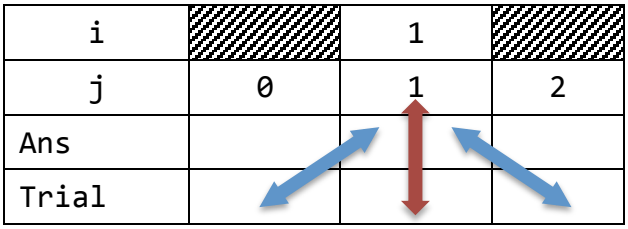
そのため、位置がずれているけれど、数値は「一致する」というのは、bite でカウントされます。それは、

```
if( ans[i] == trial[j] )      bite++;
```

で書かれています。ここにたどり着く前に、**i==j(i** と **j** の値が同じ)という条件は既に除外されています。

一つループが終わると、次は **i=1** に移ります。

次のループは、**i=1** で始まります。これに対して、**j=0, 1, 2** と変化させます。




```
for( j=0; j<3; j++ )
```

というのは、**j** を最初に **0** に設定し、３より小さい(２まで)を、１ずつ変化させて、処理を行う意味があります。

その中で、今度の

```
if( i==j ) continue;
```

は、i と j が等しい、つまり、両方とも１の時は、何もしないで次に移る、という意味に解釈します。

こうした処理を繰り返して、Eat と Bite の個数を無事、カウントし終えたら、次の１行を実行します。

```
turns++;
```

です。「何回目」というのが、１回分、増えます。

正解時の処理

callButton_Clicked で、eat == 3 の場合の処理を行いました。

```
if (eat == 3)
{
    setMessage(String.Format("おめでとう！{0} 回目で正解です！", turns));
    sp.Play();
    callButton.Enabled = false;
    messageLabel.Text = "次の問題を作成しましょう。";
}
```

sp.Play()はファンファーレの再生です。
すでに、音楽再生用の変数は宣言しました。また、音楽ファイルそのものも読み込んでいます。ですので、後は、sp.Play()だけで

音楽（効果音）が流れます。
好きなように、効果音を設定して、ゲームを組み立ててみてください。
最後の return の返却値は、checkAns として上（コールした元）に返す「要約」です。いくつ Eat で、いくつ bite だったかは、この関数が管理しています。
どんなメッセージを返すかは、設計によります。
今回は、下から上に報告書を上げる形でプログラムしました。
下から受け取った数値(eat も bite も、関数を呼び出した側でも読めます)を、上で処理して文章に仕上げる、そういう設計もできます。どうするかは、各自で考えてください。

これ以上の細かい説明は、授業中に質問があれば受け付けます。

課題

【B、C 評価の課題】

とにかく、最低限数当てゲームができるようなプログラムを仕上げてください。
最低限、各自のデスクトップを含めた実行画面があれば C 評価です。
プログラムについて理解した内容の説明などがあれば、B 評価とします。
但し、A 評価、S 評価に挑戦する人はここまでの報告は不要です。

【A、S 評価の課題】

以下の、いずれかの変更を加えてください。機能一つで、１ランク引き上げます。（10 点満点の 1 点ずつ）

(1) ヒント機能を組み込む
　例：High and Low（ボタンを一つ追加）
　ある値「以上」か「未満」かを答える　→メッセージを返す。
　自分で「ヒント」を色々と工夫したら、一つのヒントを１項目と数えます。（例：どれかの桁を教えてもらう、など）

(2) ゲーム盤全体に画像修飾を施す。（背景に画像を設定する。）など、プロパティ１項目につき 0.2 点程度（加点の上限は１点まで）

(3) 「ギブアップボタン」を作って、「降参」したら答えを教えてもらえる。

(4) 問題を４桁にしてみる。

(5) 10 回やっても正解が出せなかったら、「残念!」的な音楽を流して、答えを教えてもらう。

プログラムの説明も含めて、画面コピー、プログラムコードなどが添付されていることが条件です。ソースコードや画面などは最終形だけで構いません。

「どんな課題にするか」は自分でイメージし、自分でイメージした内容をプログラムしてみてください。
想像力を働かせて、作れそうなものに挑戦してみてください。結果がでなくても、経過が良ければ評価します。