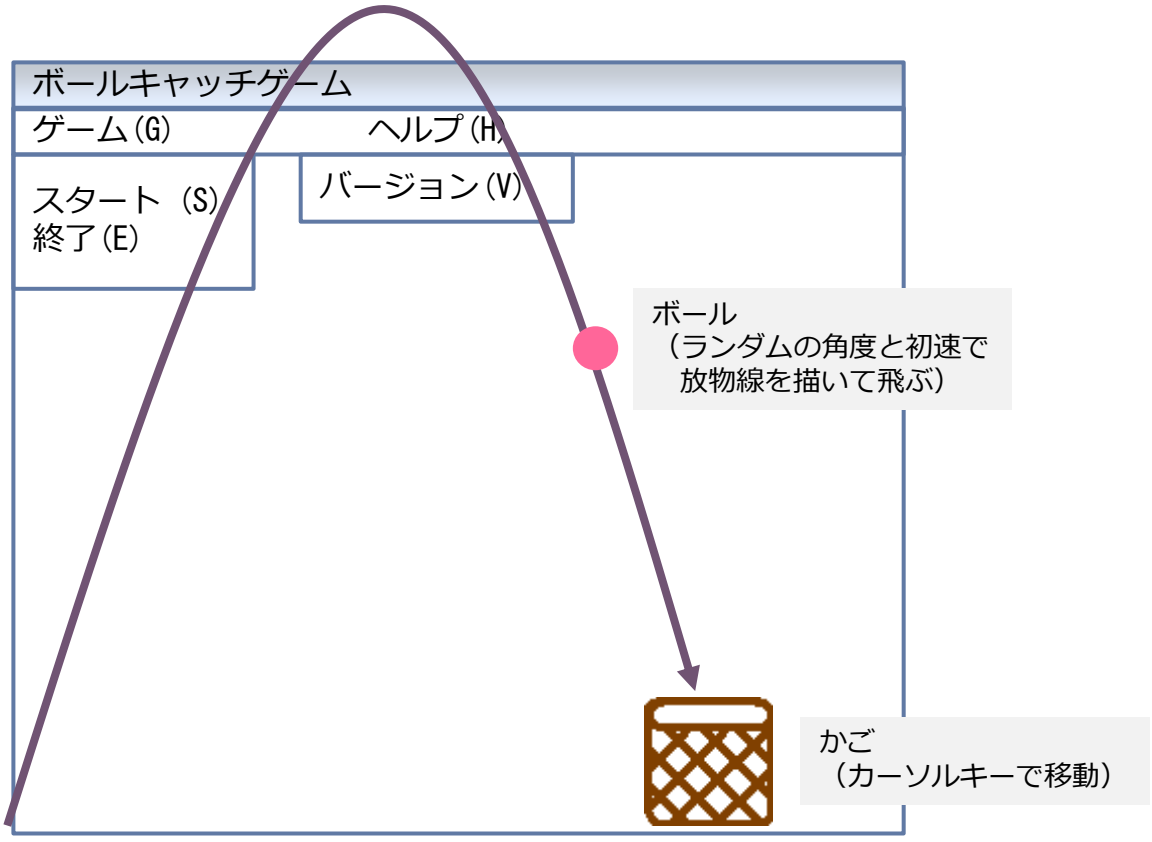
【04】 簡単なゲームを作ってみる
ボールキャッチゲーム

1 今回作成するアプリケーションの概要

放物線を描いて飛んでくるボールをかごでキャッチするプログラム

◆ 行われる動作

[1] 起動すると画面にかごとボールが表示されている
[2] メニューから「スタート」を選択すると、ゲームが開始する
「終了」を選択すると、アプリケーションが終了
「バージョン」を選択すると、バージョン情報の表示
[3] ボールは画面の向かって左側から放物線を描いて飛んでくる
[4] かごはキーボードの[→] を押すと 右に、
[←] を押すと左に移動する。
[5] ボール と かご が接触したら、かご がボールをキャッチしたことになる
ボールの位置を戻して、再び飛ばす
[6] ボール が ウィンドウ の一番下へ移動したら、キャッチ失敗でゲーム終了
[1] へ戻る



◆ 使用者とコンピュータの関係をまとめる

[使用者 → コンピュータ] メニューの「スタート」をクリック
[使用者 ← コンピュータ] ゲームスタート
= ボールを初期位置にして、タイマーがスタート

[使用者 → コンピュータ] メニューの「終了」をクリック
[使用者 ← コンピュータ] プログラムの終了

[使用者 → コンピュータ] メニューの「バージョン情報」をクリック
[使用者 ← コンピュータ] バージョン情報の表示

[使用者 → コンピュータ] キーを押す
[使用者 ← コンピュータ] どのキーが押されているかを記録
[使用者 → コンピュータ] キーが離された
[使用者 ← コンピュータ] どのキーが離されたかを記録
} ※1

[コンピュータ] タイマーが一定の時間の間隔を測定
[使用者 ← コンピュータ] ボールが移動

[コンピュータ] カーソルキーが押されている？ ※1
[使用者 ← コンピュータ] かご が移動する

[コンピュータ] ボールが かご と重なる
[コンピュータ] ボールをキャッチ
→ ボールの位置を初期位置に

[コンピュータ] ボールが 一番下へ
[コンピュータ] ボールを落とした！
→ ゲームオーバー、タイマーをストップ

※1 C# でリアルタイムにどのキーが押されているかを調べるために。
キーボードが押されたり、離されたりしたら、イベントが発生する
このときはどのキーが押されているのかを記録するだけにする
タイマーで一定時間間隔でキーが押されているときの処理を行う

◆ 必要なコントロールは次の通り
- ◆ PicuterBox コントロール（かご と ボールを表示、移動）
- ◆ Timer コントロール（一定時間間隔で処理をさせたいとき）
- ◆ MenuStrip コントロール（メニューを表示、実行）

2 ボールとかごの画像を用意する

画像はビットマップ（拡張子 .bmp）で用意する。
ボールの画像はファイル名 「**ball.bmp**」、 横 16 ドット、縦 16 ドット で用意する。

かごの画像はファイル名「**basket.bmp**」で。横 64 ドット、縦 64 ドット で用意する。

用意した画像は適当なフォルダに格納しておく。

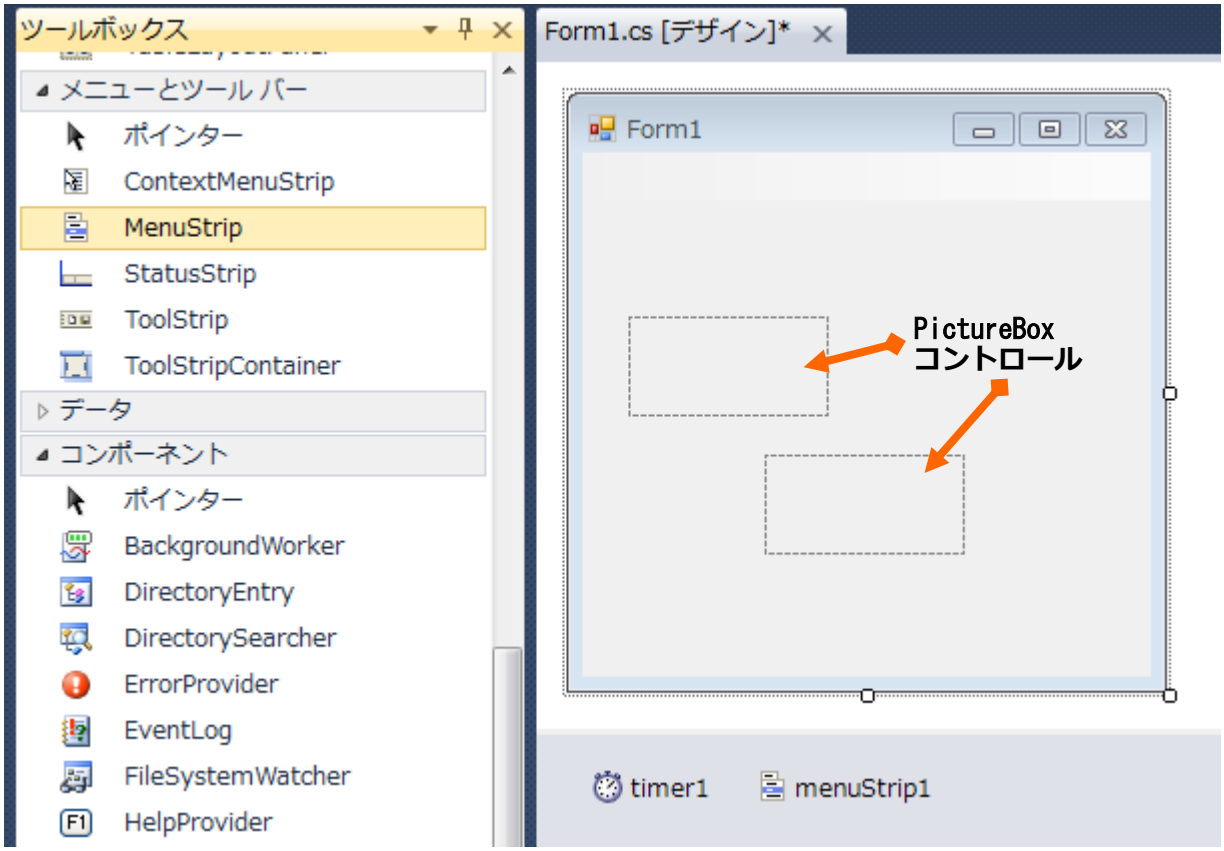
3 Visual Studio 2010 の起動と新規プロジェクトの作成

- 前回やったとおり、Visual Studio 2010 を起動
  [1] 「スタート」→ [2] 「すべてのプログラム」→ [3] 「Visual Studio 2010」のフォルダ
  → [4] 「Visual Studio 2010」のアイコン

- 新規プロジェクトも前回の手順で作成
  [1] メニュー → 「ファイル」→ [2] 「新規作成」→ [3] 「プロジェクト」
  [4]「Visual C#」→ [5] 「Windowsフォームアプリケーション」
  [6]「プロジェクト名」を入力 **（今回は JKJ04）**
  [7]「参照…」をクリックして、プロジェクトを保存する場所を選択
  [8]「ソリューションのディレクトリを作成」のチェックは はずす
  [9]「OK」をクリック

4 コントロールの配置

- **PictureBox コントロール 2 個** をツールボックスからドラッグ&ドロップして配置
- プロパティはプログラム中で設定するので、適当に配置
- **Timerコントロール**、**MenuStrip コントロール**をダブルクリックして追加する。
  （TimerコントロールもMenuStrip コントロールもフォーム上には配置されない）



■ それぞれのコントルールのプロパティを次のように設定する

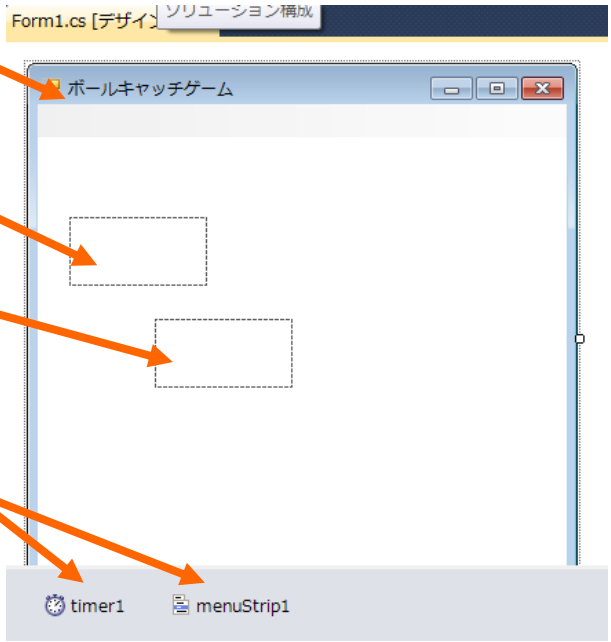
Name	Form1
KeyPreview	True
BackColor	White
Size	400, 400
Text	ボールキャッチゲーム

←（フォームでキー入力が使えるようにする）

Name	pbBall

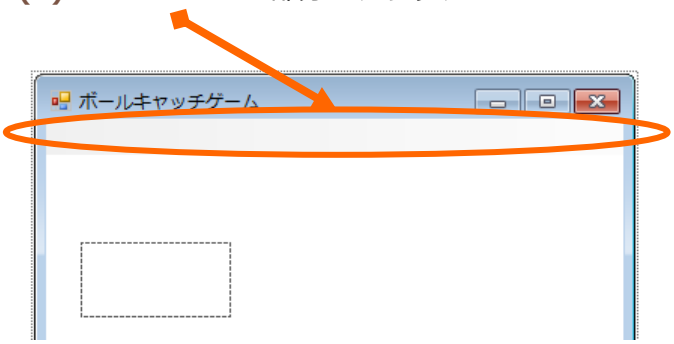
Name	pbBasket

（ 変更なし ）

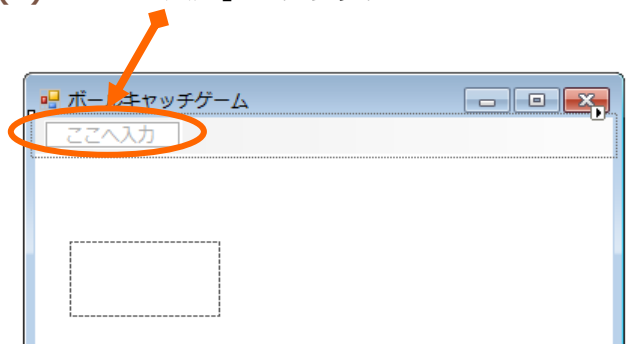


5 メニューの作成

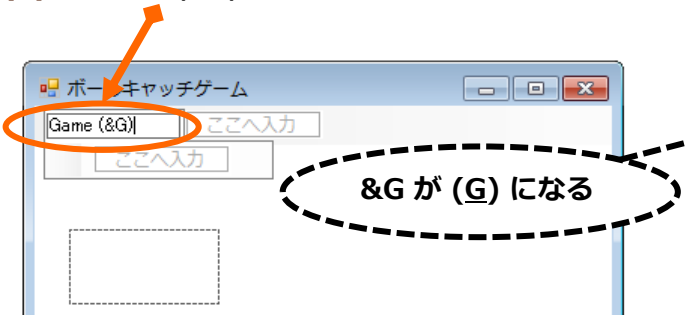
(1) メニューバーの部分をクリック



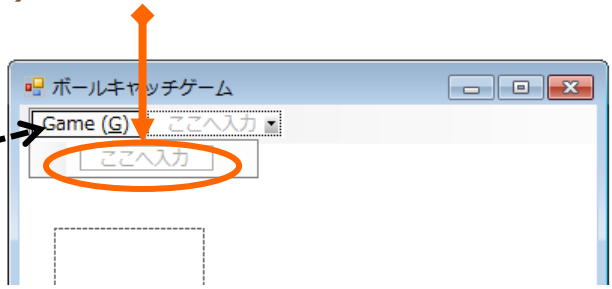
(2) 「ここへ入力」をクリック



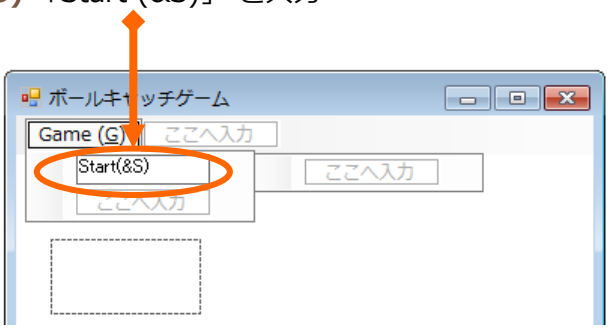
(3) 「Game (&G)」 と入力



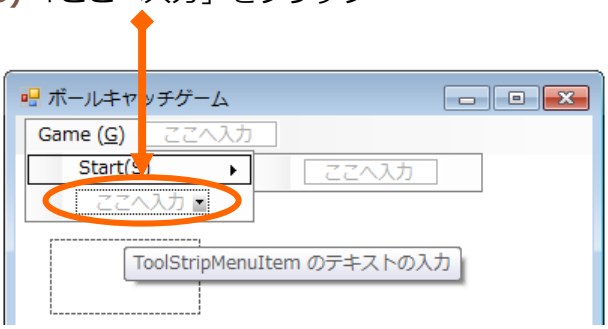
(4) 「ここへ入力」をクリック



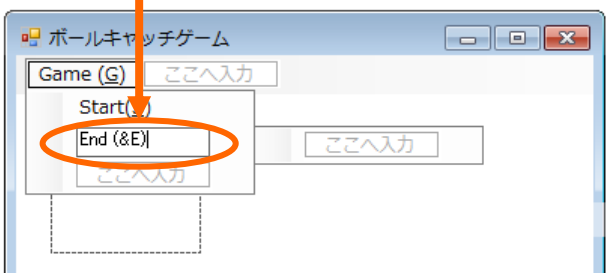
(5) 「Start (&S)」 と入力



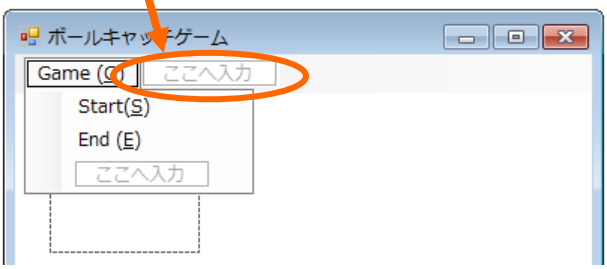
(6) 「ここへ入力」をクリック



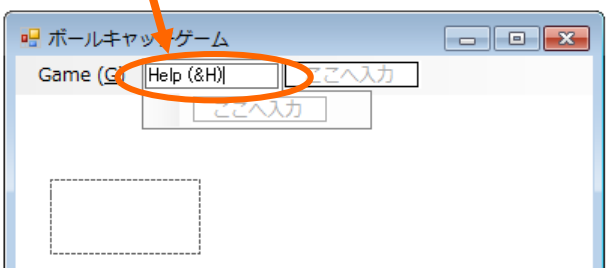
(7) 「End (&E)」 と入力



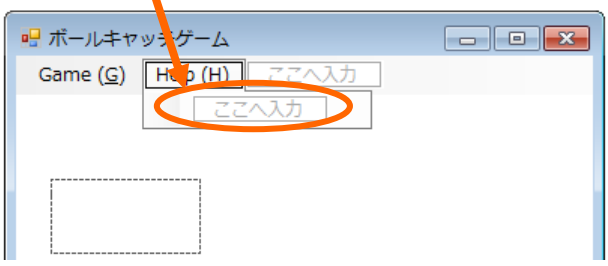
(8) 「ここへ入力」をクリック



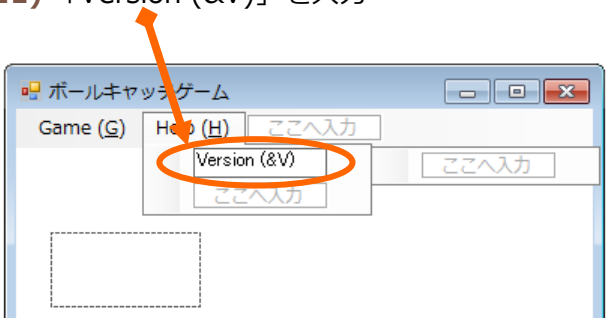
(9) 「Help (&H)」を入力



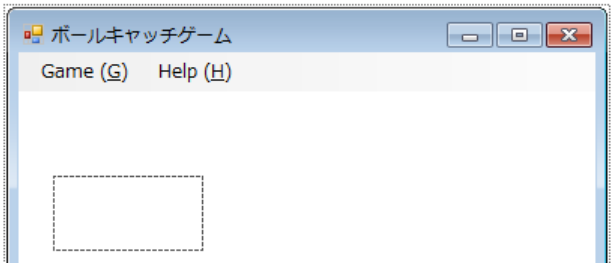
(10) 「ここへ入力」をクリック



(11) 「Version (&V)」を入力



(12) 完了



6 リソース の追加

「リソース」とは

プログラム内部にあらかじめ挿入しておくデータのこと。

文字列、イメージ（画像）、アイコン、オーディオ（音声）、ファイル、その他 の種類がある。

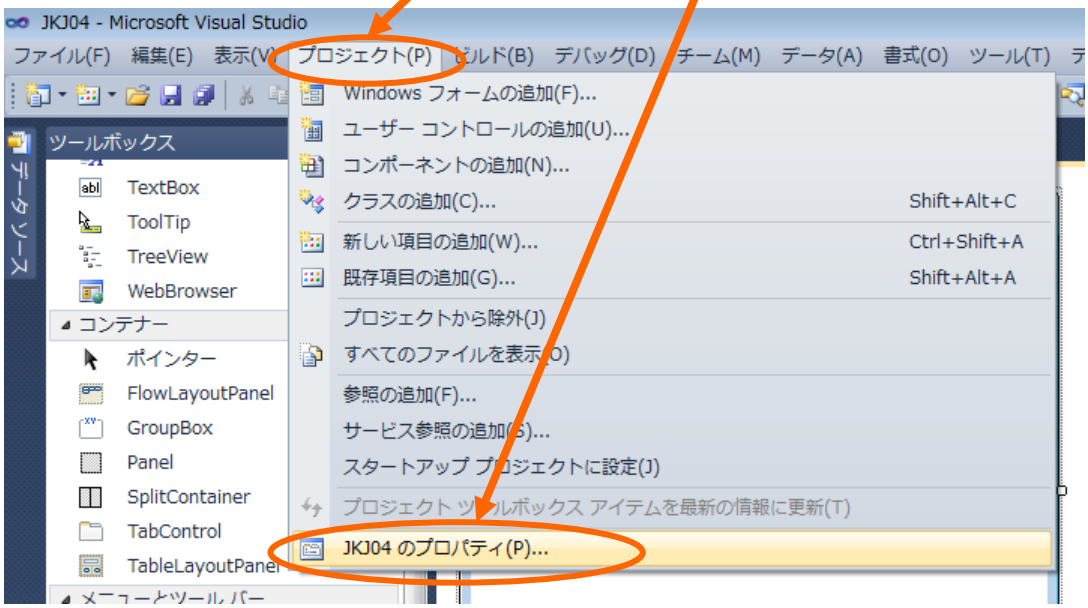
プログラム内部に取り込まれるので、プログラム自体をコピーしたり、配布するときにデータがないということが送りにくくなる。

（しかし、データを差し替えて利用することができない）

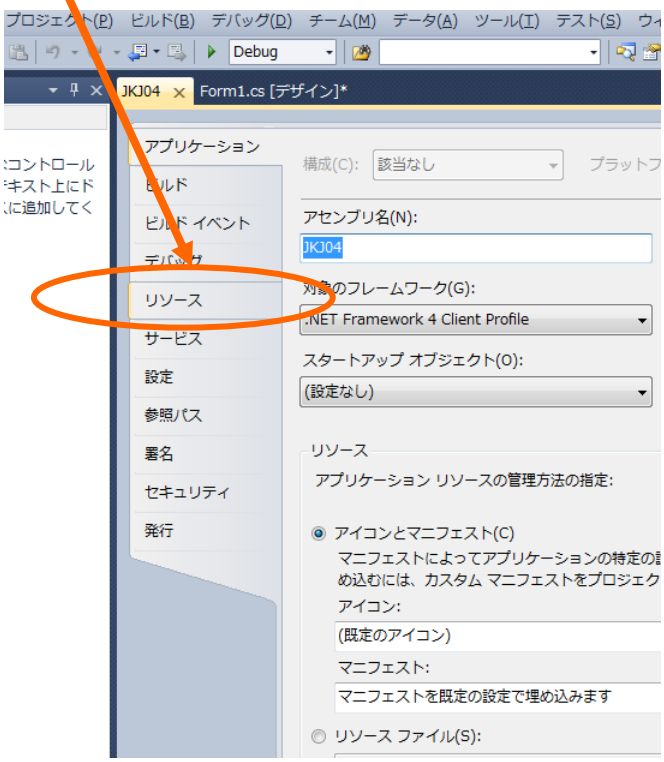
イメージ（画像）やオーディオ（音声）などを利用するとき、リソースとしてプログラム内部へ取り込むほうがいい場合と外部のファイルを利用するのがいい場合があり、用途によって使い分ける。

今回、ボールとバスケットの画像はリソースとして利用する。

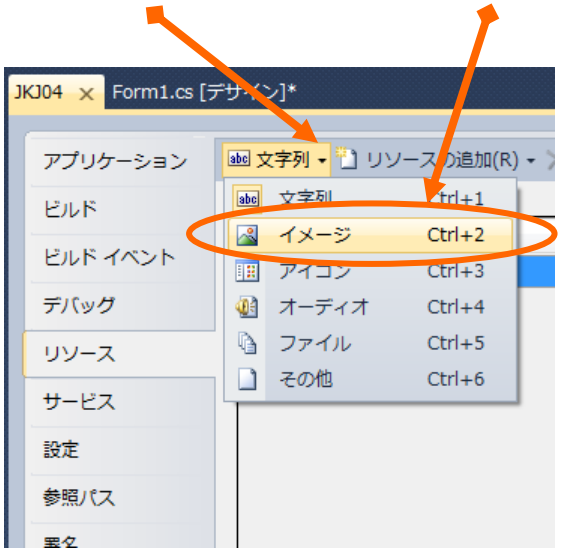
(1) Visual Studio のメニューの「プロジェクト」→「JKJ04のプロパティ…」をクリックする。



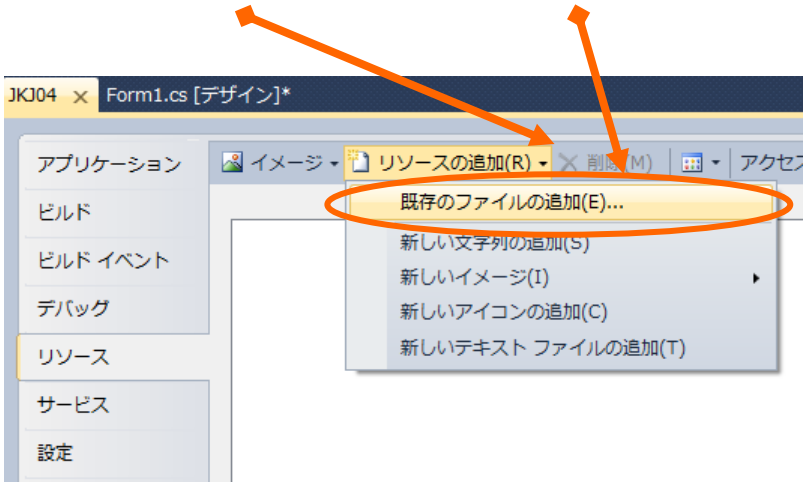
(2) 「リソース」をクリックする



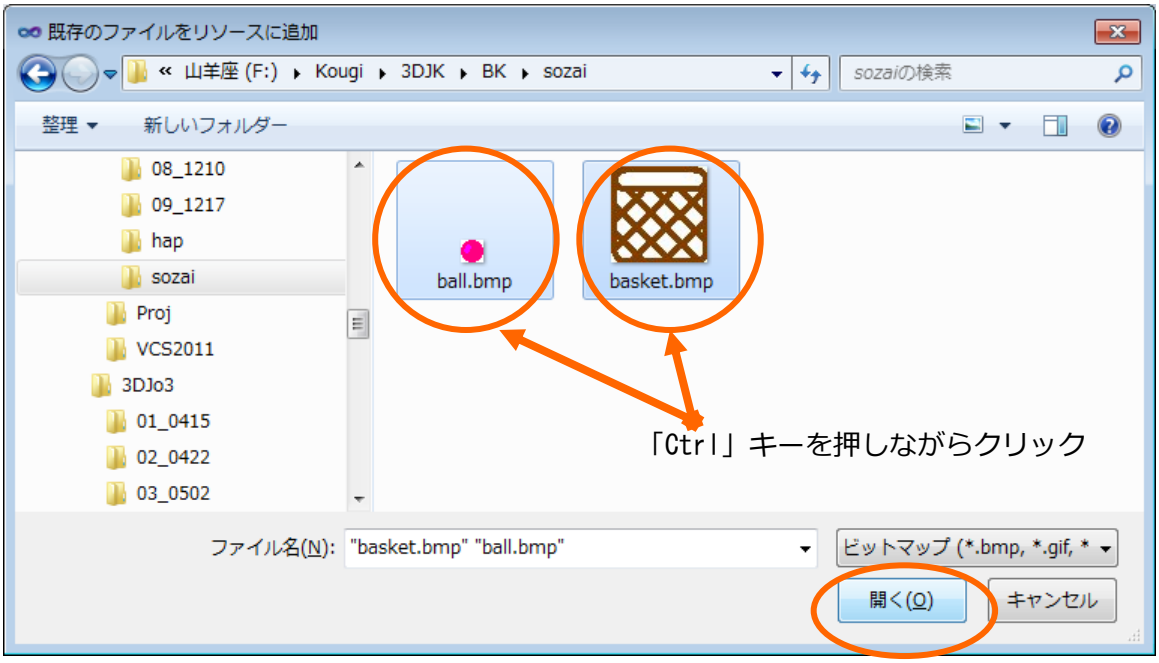
(3) **「文字列」の横の▼** をクリックして、「イメージ」をクリックする。



(4) **「リソースの追加」の横の▼** をクリックして、「既存のファイルの追加…」をクリックする。



(5) ファイルダイアログが開くので、すでに作成している（ダウンロードしている）ボールとかごの画像を選択して、「開く」をクリックする。



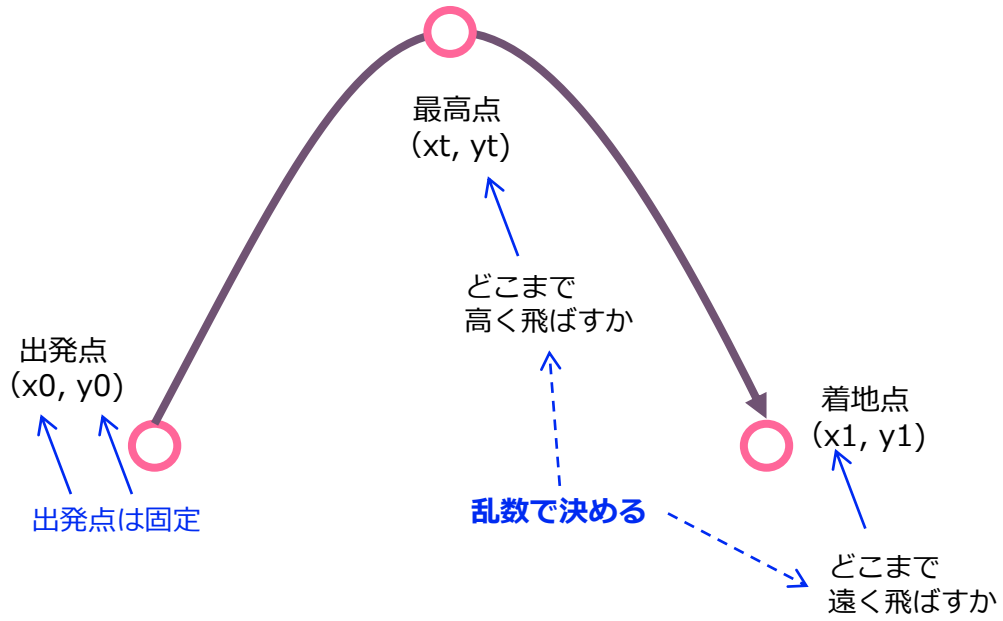
「開く」をクリック

(5) 画像のリソースへの追加が完了。
JKJ04 の 右の × をクリックして、プロパティのタグを閉じる

7 投げられるボールの動き

フォーム上へのコントロールの配置、プロパティの設定、メニューの作成、リソースの設定を行い、プログラムを入力する準備はこれで整えられた。
実際にプログラムを入力する前に、ボールがどのような動きをするのかを数式で考えておくことにする。

ボールの動きは以下の図の通り、 出発点の座標を（x0, y0）、最高点の座標を（yt, xt）、着地点の座標（x1, y1）とする。



出発点を固定し、どこまで高く飛ばすかを決める **yt** と どこまで遠く飛ばすかを決める **x1** を乱数で決定することにする。
着地点の高さを 出発点の高さと同じとすると、

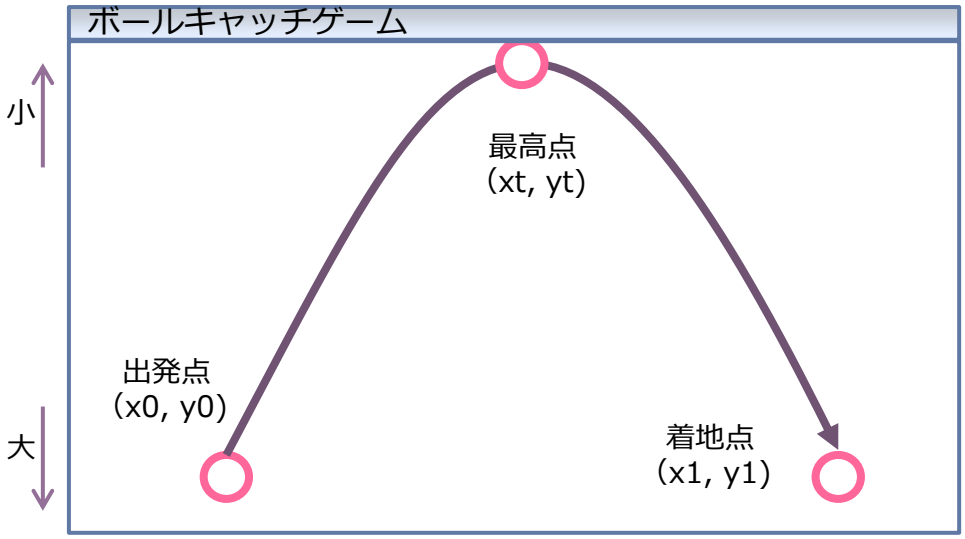
$$y1 = y0$$

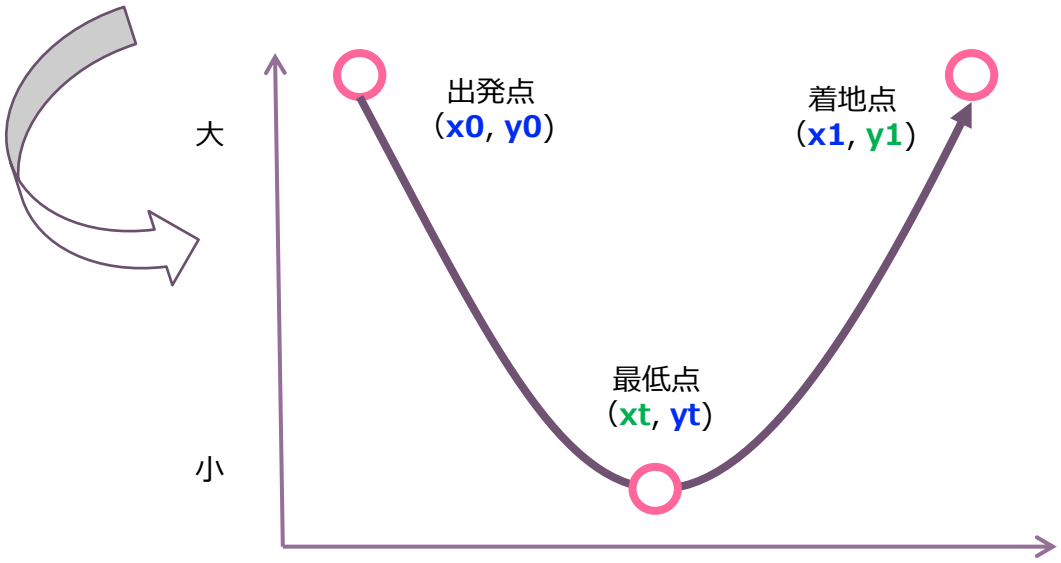
である。また、最高点の x 座標は出発点と着地点の中央であるとするなら、

$$xt = (x1 + x2) / 2$$

と表すことができる。

ボールの動きはこの 3 点を通り、最高点を（xt, yt）となる2次関数である。





しかし、フォーム上の y 座標は、数学で使う y 座標と違って上のほうが小さい。
だから数学的には上下を逆にして考える。

最小の点が (xt, yt) である 2 次関数の式は

$$y = a(x- \mathbf{xt})^2 + \mathbf{yt}$$

である。この 式に x0, y0 を与えて、 a について求めると、

$$a = (\mathbf{y0} - \mathbf{yt}) / (\mathbf{x0} - \mathbf{xt})^2$$

以上の計算より、ボールの動きを決めるアルゴリズムは、

(1) **ボールの出発点 (x0, y0) を決める**（固定）
X0 = 0, y0 = 300 にする。
y1 = y0 = 300 になる。

(2) **ボールをどれだけ高く飛ばすかを決める値 yt と どこまで遠く飛ばすかを決める値 x1 を決める。これらは乱数で決める。**
yt の 値は -300 から 200 の間の乱数とする。
負の値が範囲に入っているのは、高く飛んで見えないほうがゲームとして面白そうだからである。
x1 の値は 100 からウィンドウの幅までとする。

(3) **xt は出発点と着地点の中央になるので、次式から求める。**

$$xt = (x0 + x1) / 2$$

(4) **次の式から係数 a を求める。**

$$a = (y0 - yt) / (x0 - xt)^2$$

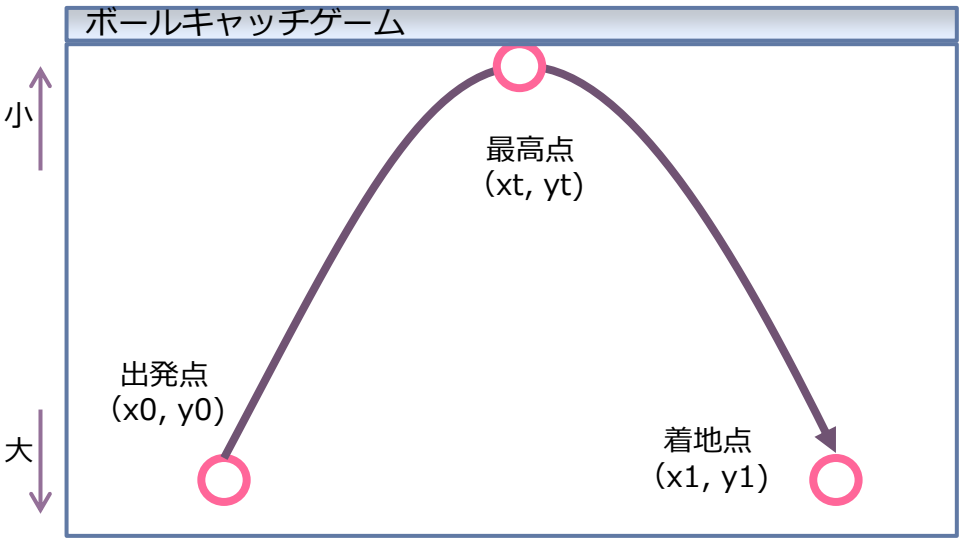
(5) よって、ボールの軌跡 (x, y) は次の式で表される。

$$y = a(x - xt)^2 + yt$$

この式を用いて x を x0 から x1 まで変化させたときの y を求めることで、ボールの放物線の動きを計算する。

ボールを投げ始める前に決める

ボールを投げている間計算する



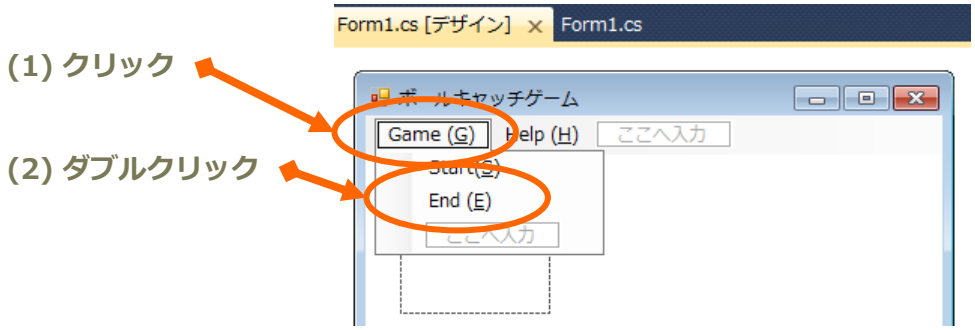
8 メニューを選択したときのプログラムを入力

■ 準備ができたので、プログラムを入力する。今回は次の順番で行う。

（1）メニューの「End(E)」をクリックしたときの動作
（2）メニューの「Version(V)」をクリックしたときの動作
（3）何かキーを押されたときの動作
（4）キーが離されたときの動作
（5）メニューの「Start(S)」をクリックしたときの動作
（6）ボールの初期位置を決める関数
（7）タイマーで一定時間ごとに動作させる処理（ボールの移動）
（8）　〃　（かごの移動）
（9）　〃　（ボールをかごで受けたときの処理）
（10）　〃　（ボールを落としたときの処理）

■ メニューの「End(E)」をクリックされたときの処理を入力

(1) Game(G)をクリックしてメニューを出す
(2) End(E)をダブルクリックしてプログラムのタグを出す



(3) プログラムを終了させる関数 Close(); を入力

自動的に入力された部分

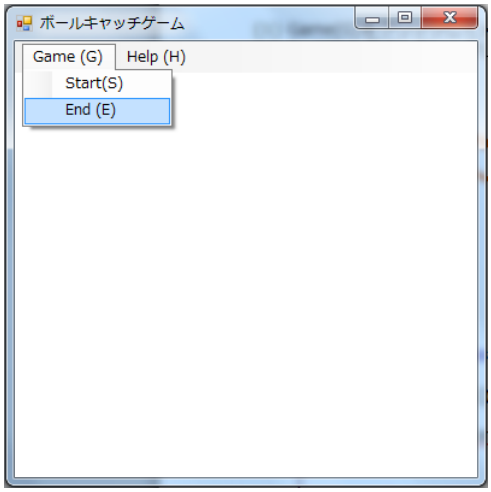
```
namespace JKJ04
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void endEToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

自動的に入力された部分

この１行を入力

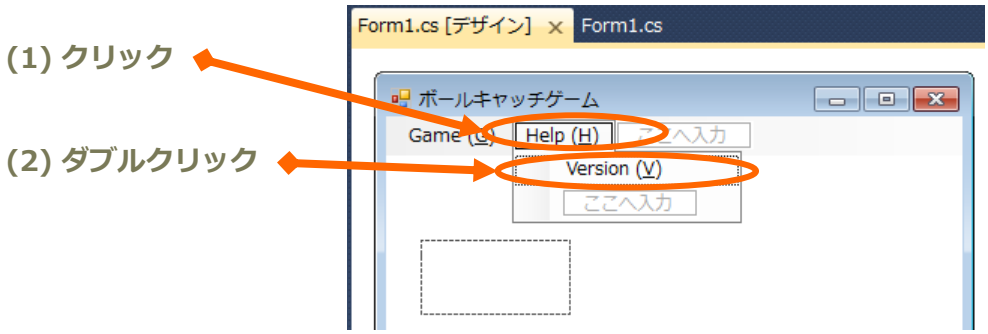
■ プログラムを実行してみる。

メニューの「End(E)」をクリックすると、プログラムが終了するか確認



■ メニューの「Version(V)」をクリックされたときの処理を入力

(1) Help(H)をクリックしてメニューを出す
(2) Version(V) をダブルクリックしてプログラムのタグを出す



(3) メッセージボックスを表示する関数 MessageBox.Show を記述

自動的に入力された部分

```
private void versionVToolStripMenuItem_Click(object sender, EventArgs e)
{
    MessageBox.Show("ボールキャッチゲーム¥n入力:占部弘治", "バージョン情報");
}
```

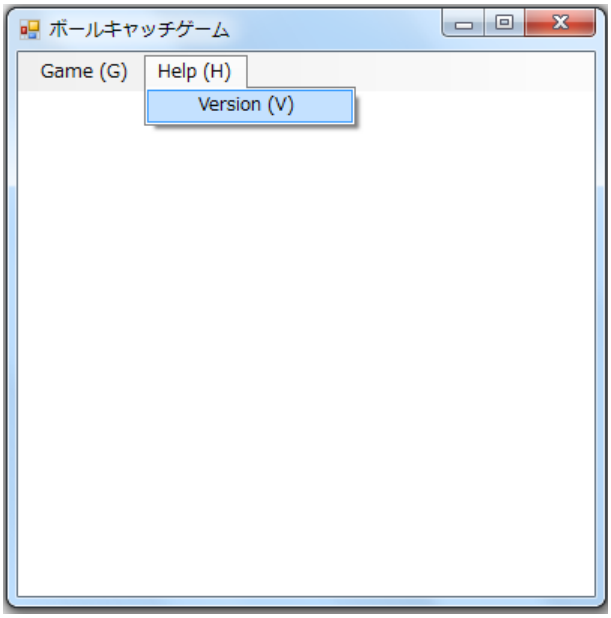
この１行を入力

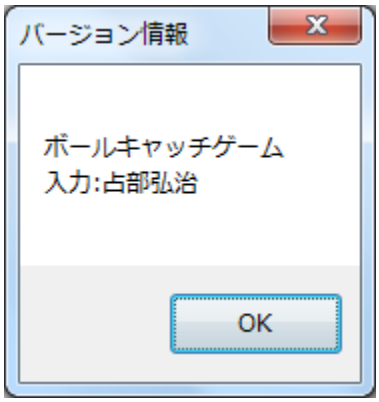
自分の名前に変更

自動的に入力された部分

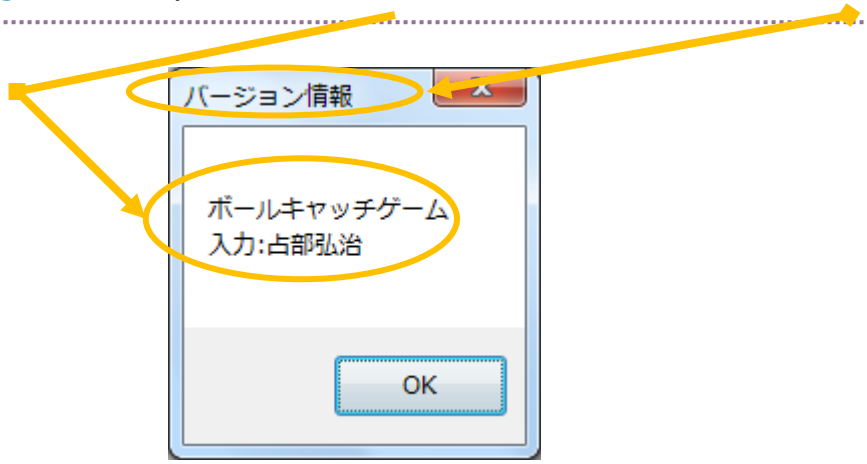
■ プログラムを実行してみる。

メニューの「Version(V)」をクリックすると、バージョン情報を掲載したメッセージボックスが表示されることを確認






```
MessageBox.Show("ボールキャッチゲーム¥r¥n入力:占部弘治", "バージョン情報");
```

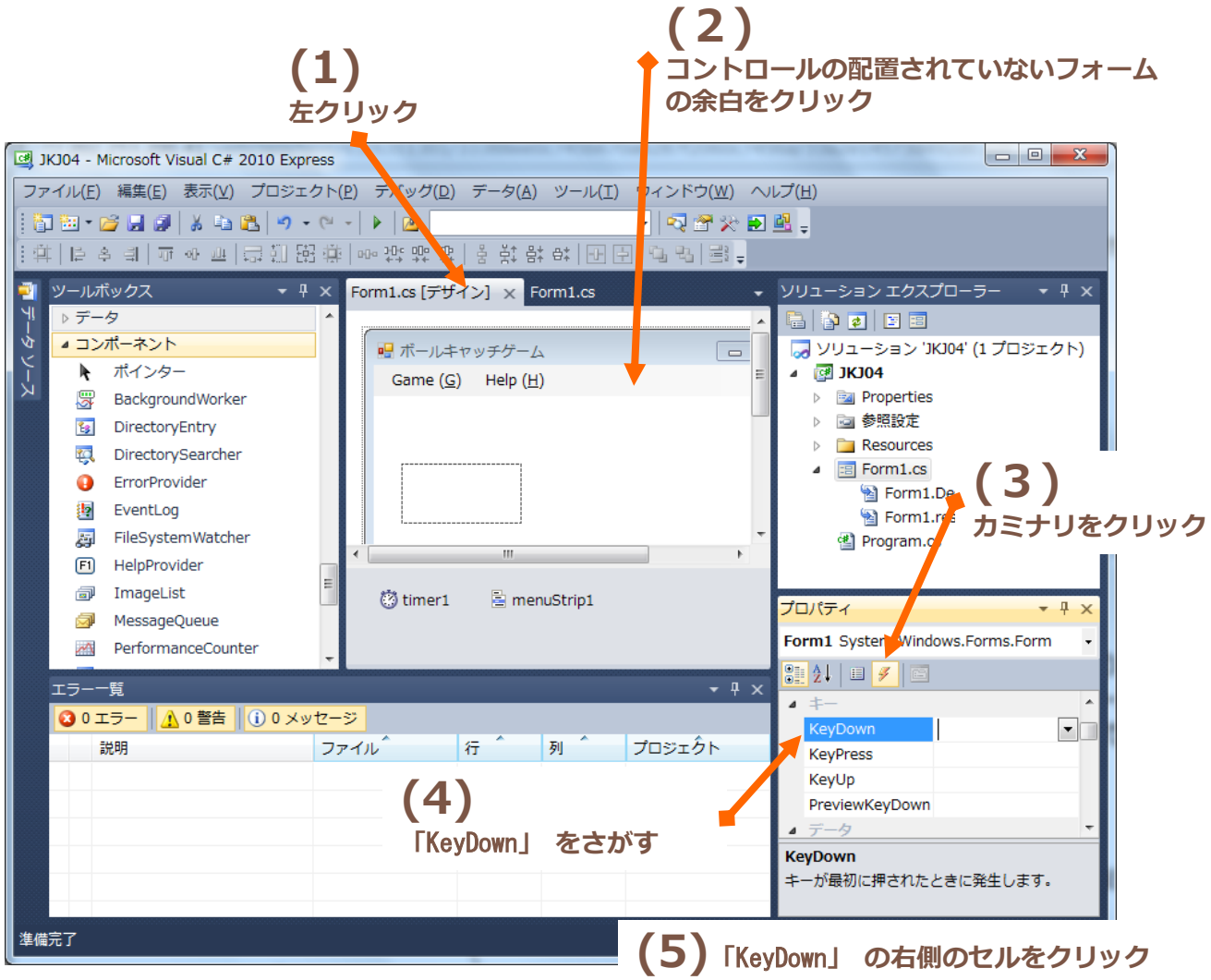


小さいウィンドウ（メッセージボックス, MessageBox）が表示される。
1番目の引数がウィンドウに、2番目の引数がタイトルバーに表示

改行は ¥r¥n

9 キーを押されたときのプログラムを入力

- キーを押されたときの生じるイベント KeyDown が発生したとき呼び出される関数を生成
  - (1) Form1.cs [デザイン] をクリックして、コード入力のタグからフォームのデザインするタグへ戻る
  - (2) From1 のプロパティを選ぶ。（フォームのコントロールの配置してない場所をクリック）
  - (3) プロパティウィンドウのカミナリのところをクリックしてイベント一覧を表示
  - (4) イベント一覧の中から「KeyDown」を探す
  - (5) KeyDown の右側のテーブルをクリックしてカーソルが点滅した状態にする
  - (6) [Enter] キーを押す



(6) 押されたキーの保持されるよう次のプログラムを入力

```
namespace JKJ04
{
    public partial class Form1 : Form
    {
        bool[] PressKey = new bool[256];

        public Form1()
        {
            InitializeComponent();

            for (int i = 0; i < PressKey.Length; i++)
            {
                PressKey[i] = false;
            }
        }

        private void endEToolStripMenuItem_Click(object sender, EventArgs e)
        {

            Close();

        }

        private void versionVToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("ボールキャッチゲーム¥n入力:占部弘治", "バージョン情報n");
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyValue < PressKey.Length)
            {
                PressKey[e.KeyValue] = true;
            }
        }
    }
}
```

押されたキーを保存する配列を用意する

この１行を入力

押されたキーを保存する配列を初期化

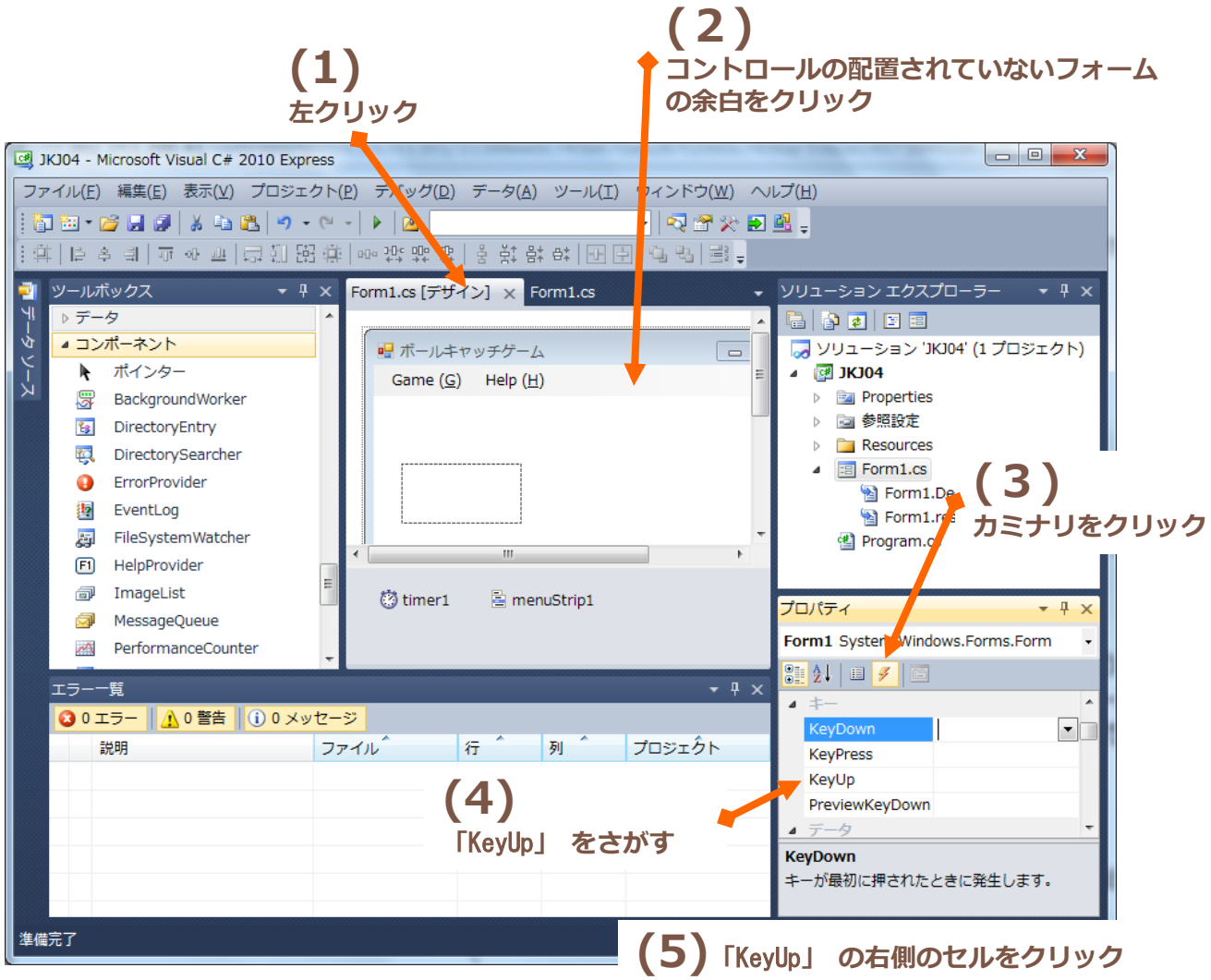
この４行を入力

押されたキーの配列番号のところを true に

この４行を入力

１０ キーが離されたときのプログラムを入力

- キーを押されたときの生じるイベント KeyUP が発生したとき呼び出される関数を生成
  - (1) Form1.cs [デザイン] をクリックして、コード入力のタグからフォームのデザインするタグへ戻る
  - (2) From1 のプロパティを選ぶ。（フォームのコントロールの配置してない場所をクリック）
  - (3) プロパティウィンドウのカミナリのところをクリックしてイベント一覧を表示
  - (4) イベント一覧の中から「KeyPUp」を探す
  - (5) KeyUp の右側のテーブルをクリックしてカーソルが点滅した状態にする
  - (6) [Enter] キーを押す



(6) 押されたキーの保持されるよう次のプログラムを入力

```
private void Form1_KeyUp(object sender, KeyEventArgs e)
    {
        if (e.KeyValue < PressKey.Length)
        {
            PressKey[e.KeyValue] = false;
        }
    }
```

押されなくなったキーの配列番号のところを true に

この４行を入力

１１ 全体で使う変数や乱数などを宣言する

- メンバ変数を宣言する部分に変数の宣言と乱数のインスタンスの生成を記述する

自動的に入力された部分

```
public partial class Form1 : Form
    {
```

入力済

```
    bool[] PressKey = new bool[256];
```

```
    double x0, y0;  // 出発点の座標
    double xt, yt;  // 最高点の座標
    double x1, y1;  // 着地点の座標
    double a;       // 係数

    // 乱数を発生させるクラス Random から
    //          インスタンス rand を生成
    Random rand = new Random();
```

この部分を入力

```
    public Form1()
    {
        InitializeComponent();
```

自動的に入力された部分

- プログラム起動時に実行される部分（コンストラクタ）を入力する
  ここではプログラム起動時にリソースの画像を PictureBox へ収納している

```
public Form1()
{
    InitializeComponent();

    for (int i = 0; i < PressKey.Length; i++)
    {
        PressKey[i] = false;
    }

    // 画像をリソースから参照する。
    // 画像の大きさ（Size）もリソース画像の大きさ（Size）から
    pbBall.Image = Properties.Resources.ball;
    pbBall.Size = Properties.Resources.ball.Size;
    pbBasket.Image = Properties.Resources.basket;
    pbBasket.Size = Properties.Resources.basket.Size;
```

この部分を入力

```
private void endEToolStripMenuItem_Click(object sender, EventArgs e)
{

    Close();

}
```

自動的に入力された部分、
または以前に入力している部分

12 ボールの位置を初期化する関数

- ボールをどこまでの高く、どこまで遠く飛ばすかを乱数で決定し、
  その他の値を計算する部分
  ゲーム開始時とボールをキャッチした時に実施されるので関数にしておく。

入力済

```
Random rand = new Random();
```

```
void InitBall()
{
    // 出発点の座標
    x0 = 0.0; y0 = 300.0;

    // ボールの場所を設定
    pbBall.Left = (int)x0;
    pbBall.Top = (int)y0;
    pbBall.Visible = true;

    // 着地点の座標
    x1 = (double)rand.Next(100, this.Width-pbBall.Width);
            // 100 から ウィンドウの端までをランダムに
    y1 = y0;

    // 最高点の座標
    xt = (x0 + x1) / 2.0;
    yt = (double)rand.Next(500) - 300.0;
            // 最小 -300 から 最大 200 までをランダムに

    // 係数の計算
    a = (y0 - yt) / ((x0 - xt) * (x0 - xt));

}
```

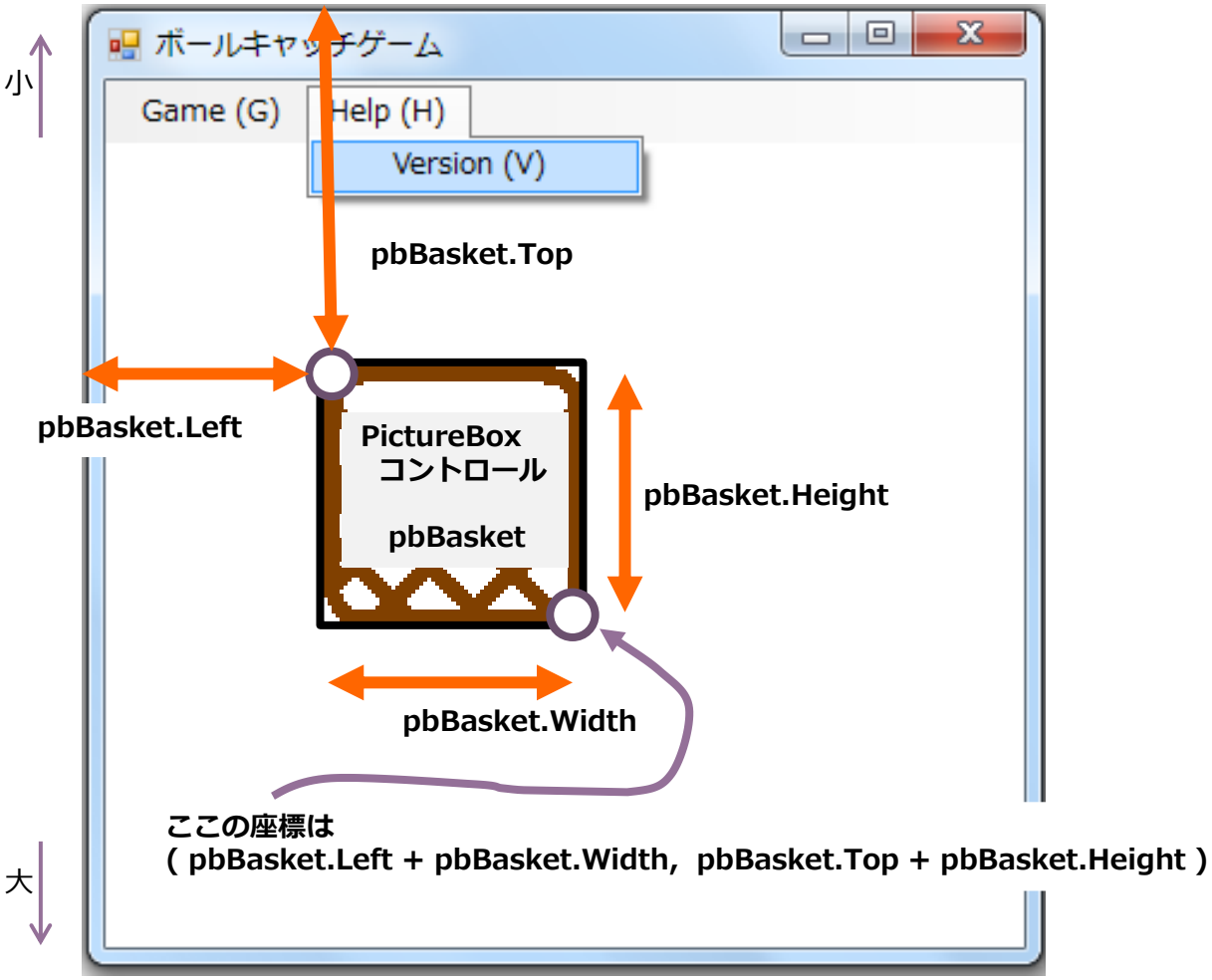
この部分を入力

```
public Form1()
{
    InitializeComponent();
```

自動的に
入力されている
部分

■フォーム上でのコントロールの配置

座標の原点は右上
下に行くほど値が大きくなる（数学のグラフと逆向き）



■乱数とは

発生する数字の前後に関連性はないでたらめな順番で
ならぶ数列のこと

乱数を利用することで、発生する事象に規則性がなくなり、
ゲームのコンピュータ側の動きが偶然になる

■乱数の使い方

乱数を発生させるクラス Random のインスタンス を生成

```
Random rand = new Random();
```

（乱数の種には現在の時刻を用いている）

※ 乱数の種とは乱数の数列を発生させる元の数字
乱数の種が同じなら同じ数列の乱数が発生する
→ 乱数の種はプログラムが実行されたとき １回だけ与える

生成したインスタンス rand のメンバ関数 Next () を使うと
乱数が得られる。

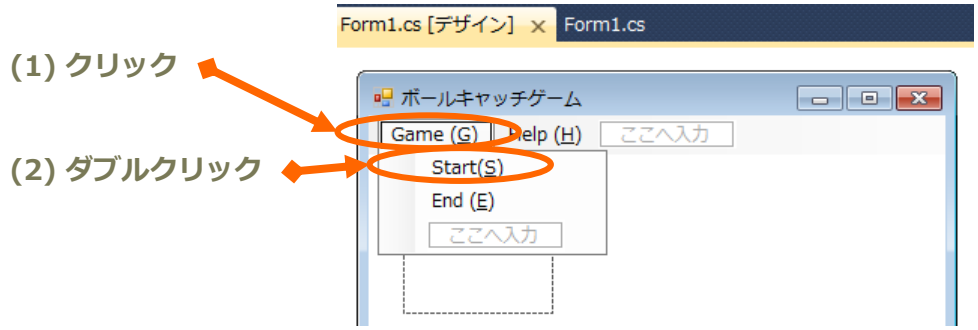
例：

```
A = rand.Next( );
    // 0 以上 の乱数（0 ≦ A < 0x0FFFFFF）
B = rand.Next(300);
    // 0 以上 300 未満（0 ≦ B < 300）の乱数を発生
C = rand.Next(100,300);
    // 100 以上 300 未満 の乱数を発生
```

１３ ゲーム開始時の処理

- メニューの「Start(S)」をクリックされたときの処理を入力

(1) Game(G)をクリックしてメニューを出す
(2) Start(S)をダブルクリックしてプログラムのタグを出す



(3) ゲーム開始時に必要な処理を入力する

自動的に入力された部分

```
private void startSToolStripMenuItem_Click(object sender, EventArgs e)
        {
```

```
// ボールの位置を初期化する
InitBall();

// かご の位置も初期化する
pbBasket.Top = 300;
pbBasket.Left = 200;

// タイマーの設定
timer1.Interval = 20;
timer1.Start();
```

この部分を入力

```
        }
```

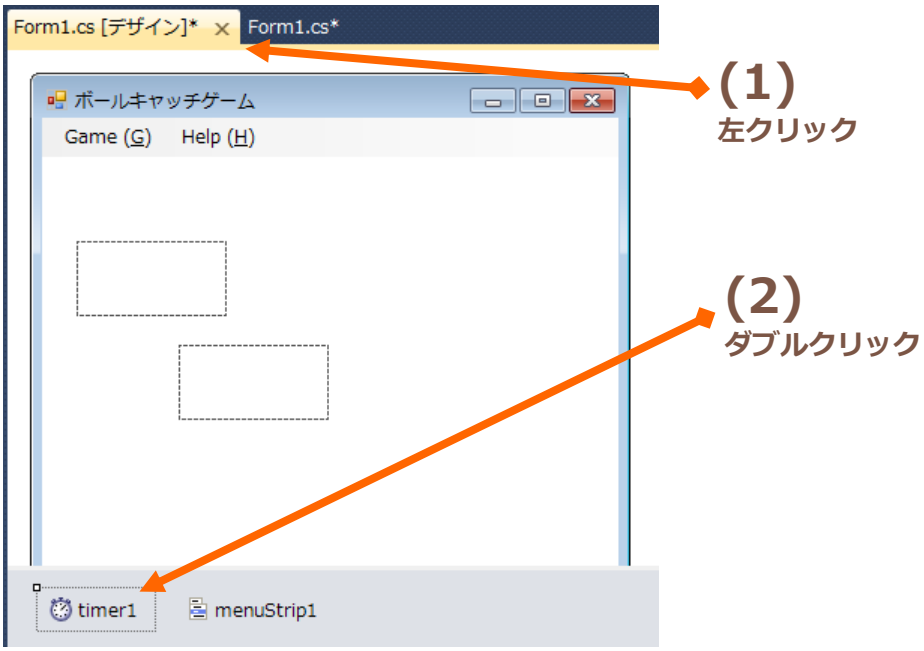
自動的に入力された部分

１４ タイマーが動作したときの処理

タイマーが動作し、一定の時間間隔で以下のことが処理されることで、ゲームが進行する。

（７）ボールの移動（計算）
（８）かごの移動（キーボードの入力から計算）
（９）ボールをかごで受けたときの処理（当たり判定）
（１０）ボールを落としたときの処理（

(1) Form1.cs [デザイン] をクリックして、コード入力のタグからフォームのデザインするタグへ戻る
(2) timer1 をダブルクリック



(3) ゲームに必要な処理を入力する

```
private void timer1_Tick(object sender, EventArgs e)
    {
```

自動的に入力された部分

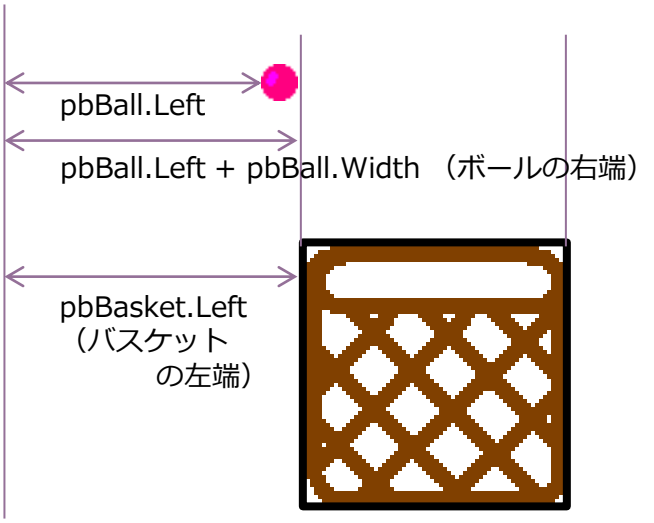
```
// ボールの移動
pbBall.Left += 1;
double x = (double)pbBall.Left;
pbBall.Top = (int)(a * (x - xt) * (x - xt) + yt);

// かごの移動（左）
if (PressKey[(int)Keys.Left] == true)
{
    if (pbBasket.Left > 100)
    {
        pbBasket.Left -= 2;
    }

}

// かごの移動（右）
if (PressKey[(int)Keys.Right] == true)
{
    if (pbBasket.Left < this.Width)
    {
        pbBasket.Left += 2;
    }
}

// 当たり判定
if ((pbBall.Top + pbBall.Height > pbBasket.Top) &&
    (pbBall.Top < pbBasket.Top + 20) &&
    (pbBall.Left + pbBall.Width > pbBasket.Left) &&
    (pbBall.Left < pbBasket.Left + pbBasket.Width))
{
    // ボールが かご に入っていたら…
    InitBall(); // ボールを初期状態に
}

// ボールがフォーム⊞り下になったかどうかを判定
if (pbBall.Top > this.Width)
{
    // ボールを落としたとして、ゲーム終了
    timer1.Stop(); // タイマー停止
}
```

この部分を入力

```
}
```

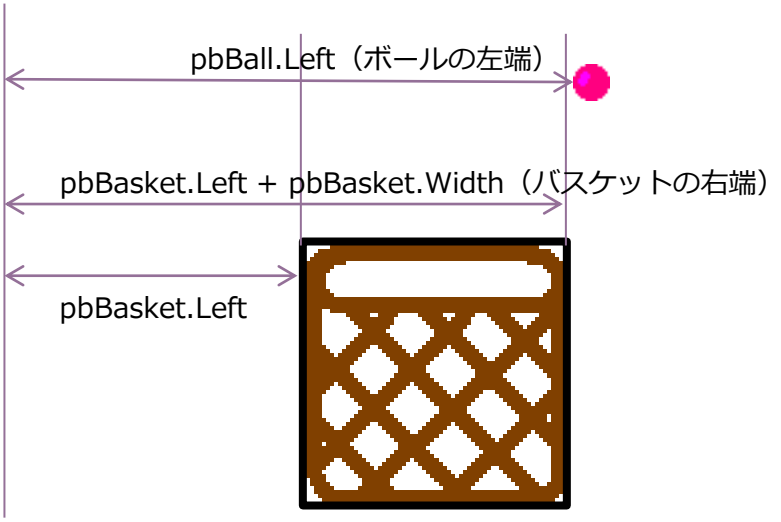
自動的に入力された部分

１５ 当たり判定 の説明



pbBall.Left + pbBall.Width > pbBasket.Left
ボールの右端　　バスケットの左端

ボールの右端 が バスケットの左端 より 右にある



この両方が成立しているとき、
ボールと かご が横方向に
重なっているとする

pbBall.Left < pbBasket.Left + pbBasket.Width
ボールの左端　　バスケットの右端

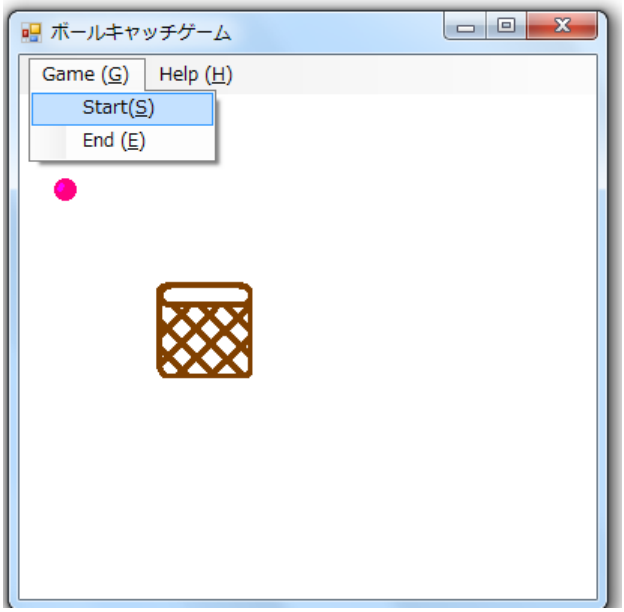
ボールの左端 が バスケットの右端 より 左にある

縦方向も同様に考える。
pbBall.Top + pbBall.Height > pbBall.Top
pbBall.Top < pbBasket.Top + pbBasket.Height

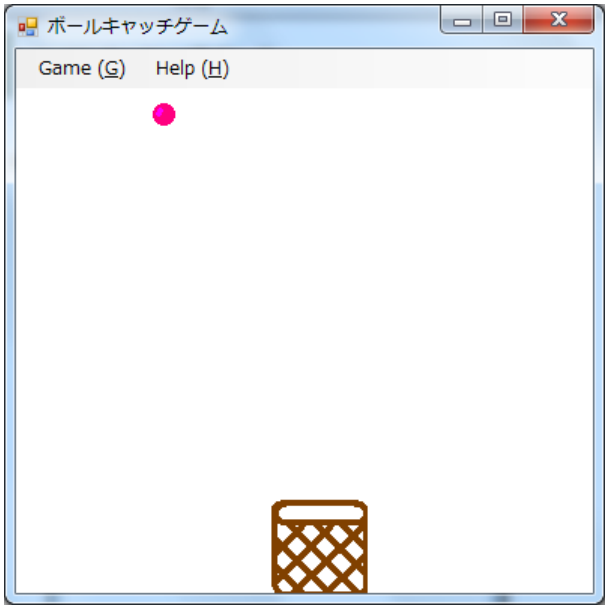
この４つの条件がすべて成立していれば、ボールと かご が当たっていると考える

１６ とりあえず完成

- メニューの「Start(S)」をクリックすると、ゲーム開始



- 右下からボールが放物線を描いて飛んでくる
- キーボードの「←」「→」のキーを押して、駕籠を移動



１７ まとめ

- メニューを使う
  - MenuStrip コントロールを利用
  - クリックしてメニューのタイトルを入力すれば、自在に作成できる
  - タイトルのところをダブルクリックして、イベントで呼び出される関数へ

- リソースを使う
  - リソースとはプログラムに埋め込むデータ（イメージ、オーディオなど）
  - データはあらかじめ用意する。ファイル名がリソース名になるので注意
  - 「プロジェクト」→「プロパティ」→「リソース」の順にクリック
  - 画像を追加するときは、「文字列」を「イメージ」に変更
  - あらかじめ用意したファイルを使うときは、
    「リソースの追加」▼の▼の部分をクリックして、
    「既存のファイルの追加…」を使う
  - プログラム中では
    Properties.Resources.（リソース名）で参照できる

- キー入力をつかう
  - フォームのプロパティ KeyPreview を True にする
  - KeyDown イベント と KeyUp イベントでどのキーが押されたかをチェックする
  - どのキーが押されたかをチェックするには キーの数の配列を用意する
  - Timer などの処理でこの配列をチェックすれば、どのキーが今押されているかわかる

- MessageBox.Show(ウィンドウに表示する文字列, タイトルに表示する文字列）; で
  小さいウィンドウでメッセージを表示することができる

- 今回使ったコントロール
  - PictureBox コントロール（画像を表示したり、描画を管理したり）
  - Timer コントロール（一定の時間間隔でイベントを発生）
  - MenuStrip コントロール（メニューを管理する）**<- NEW!**

１８ 追加課題

1 ボールをキャッチしたら、得点が増えるようにし、その得点を表示する。

2 タイトルとゲームオーバーの画面を作る

3 2個以上のボールが飛んでくるようにする。
同時に着地すると絶対に捕れないので、捕れるように時間差をつけること。