[OpenGL 2D 2018 第04回]

ダンジョンでじゃんけんをするのは  
間違っているだろうか

# ダンジョンのデータ構造

## ダンジョンのデータ化

今回はダンジョン探索機能を追加しようと思います。  
今回の講義からプログラムを書く行数が増えていますので、

まずは、プログラムでダンジョンを表現する方法を考えましょう。どんなものであれ、コンピューターに扱わせようと思うなら、何らかのデータに落とし込まなければなりません。

簡単のために、ダンジョンには通路と壁しか存在しないこととします。また、ある地点には壁がないか壁があるかのどちらかしかあり得ないとします。そうすると、例えば壁がない場合は0、壁がある場合は1というようにデータ化できそうです。

さらに、ダンジョン全体を碁盤の目のように四角いマスで区切ります。この各マスについて、壁があるかないかを示すことができれば、ダンジョンをデータ化することができそうです。

それでは、最初にダンジョンの大きさを決めましょう。  
cpu\_win\_count変数の定義の下に、次のプログラムを追加してください。

int cpu\_win\_count = 0; **+**// ダンジョンの大きさ  
**+**const int dungeon\_width = 8;  
**+**const int dungeon\_height = 8;

dungeon\_width(だんじょん・うぃす)がダンジョンの横のマス目の数、dungeon\_height(だんじょん・はいと)が縦のマス目の数です。この2つを配列の大きさとしてダンジョンのデータを作ると、次のようになります。dungeon\_height変数の定義の下に、次のプログラムを追加してください。

const int dungeon\_height = 8;  
  
**+**/\*\*  
**+**\* ダンジョンマップ.  
**+**\*  
**+**\* 0: 通路  
**+**\* 1: 壁  
**+**\*/  
**+**const char dungeon\_map[dungeon\_width][dungeon\_height] = {  
**+**{ 1, 1, 1, 1, 1, 1, 1, 1 },  
**+**{ 1, 0, 1, 0, 0, 0, 0, 1 },  
**+**{ 1, 0, 1, 1, 1, 0, 1, 1 },  
**+**{ 1, 0, 0, 0, 1, 0, 0, 1 },  
**+**{ 1, 0, 1, 0, 1, 1, 0, 1 },  
**+**{ 1, 1, 1, 0, 0, 0, 0, 1 },  
**+**{ 1, 0, 0, 0, 1, 0, 1, 1 },  
**+**{ 1, 1, 1, 1, 1, 1, 1, 1 },  
**+**};

ご覧のとおり、壁のない部分(通路)を0、壁のある部分を1とする簡単な迷路になっています。  
このマップは横がX座標(右方向がプラス)、縦がY座標(下方向がプラス)、左上が原点(0,0)となります。画面の座標とは違うので、混乱しないように気をつけてください。

ところで、迷路にはスタートとゴールが必要ですね。dugeon\_map変数の定義の下に、次のプログラムを追加してください。

{ 1, 1, 1, 1, 1, 1, 1, 1 },  
 };  
  
**+**// スタートの位置.  
**+**const int start\_x = 1;  
**+**const int start\_y = 1;  
**+**  
**+**// ゴールの位置.  
**+**const int goal\_x = 3;  
**+**const int goal\_y = 1;

start\_xとstart\_yはダンジョンマップ内のスタート地点の座標、goal\_xとgoal\_yは同様にゴール地点の座標になっています。これでダンジョンのデータ化は完了です。

## プレイヤーのデータ化

紙の上の迷路なら、あとは手と鉛筆があれば遊ぶことができるでしょう。しかし、コンピューターにはそんな便利なものは備わっていないので、プログラムを作る人が代わりになるものを用意してあげます。  
まずはプレイヤーの位置を現す変数を作りましょう。goal\_y変数の定義の下に、次のプログラムを追加してください。

const int goal\_y = 1;  
  
**+**// プレイヤーの現在位置.  
**+**int player\_x;  
**+**int player\_y;

手と鉛筆の代わりに、この変数でダンジョン内の現在位置を記録します。  
もうひとつ、プレイヤーの向きをあらわす変数も作りましょう。player\_y変数の定義の下に、次のプログラムを追加してください。

int player\_y;  
  
**+**// プレイヤーの向き.  
**+**const int dir\_up = 0;  
**+**const int dir\_right = 1;  
**+**const int dir\_down = 2;  
**+**const int dir\_left = 3;  
**+**int player\_direction;

ダンジョンは四角いマスで区切られているので、プレイヤーの向きもマス目の方向に合わせて4方向とします。つまり0から3の値を方向に割り当てればいいのですが、数値では分かりにくいですよね。そこで、各方向を示す変数を作ることにしました。それがdir\_up(でぃあ・あっぷ)、dir\_right(でぃあ・らいと)、dir\_down(でぃあ・だうん)、dir\_left(でぃあ・れふと)です。数値が1増えるごとに時計回りに方向が変わります。  
そして、player\_direction(ぷれいやー・でぃれくしょん)がプレイヤーの向きです。上記の4つのうちいずれかの値が設定されることになります。

# ダンジョンを描く

## プレイヤーの初期設定

タイトル画面が終わってゲームを開始する前に、プレイヤーの初期設定をします。  
タイトル画面のプログラムの最後に、次のプログラムを追加してください。

reset\_all\_text();  
 reset\_image(1);  
 fade\_in(1);  
**+**  
**+** // ゲームの初期設定を行う.  
**+** player\_x = start\_x;  
**+** player\_y = start\_y;  
**+** player\_direction = dir\_down;  
**+** aiko\_flag = false;  
 }

## 文字と画像を消す

これからダンジョンを表示するわけですが、そのまえに文字と画像を消しておきましょう。  
タイトル画面のプログラムの後ろに、次のプログラムを追加してください。

player\_direction = dir\_down;  
 aiko\_flag = false;  
 }  
**+  
+**/\*  
**+**\* ダンジョン探索.  
**+**\*/  
**+**reset\_all\_image();  
**+**reset\_all\_text();

## 壁の位置を調べる

ダンジョンといえば3Dダンジョンですよね。  
3Dダンジョンを表示するには、プレイヤーが向いている方向を基準として、左右と前方のどの位置に壁があるかが分かれば十分です。あとは壁がなければなにも表示せず、壁があればその位置に壁の画像を表示するだけです。

どのくらいの範囲の壁を調べればいいかは、どのくらいの範囲を見えるようにするかによります。今回は、プレイヤーの左右と正面、左前方、右前方の5マスを調べることにします。  
まずは調べた結果を格納する変数を定義します。2.1で追加したプログラムの下に、次のプログラムを追加してください。

/\*  
 \* ダンジョン探索.  
\*/  
reset\_all\_image();  
reset\_all\_text();  
**+  
+**// 方向に応じて壁の有無を調べる.  
**+**int wall\_left[2] = {};  
**+**int wall\_right[2] = {};  
**+**int wall\_front[1] = {};

左側2マス、右側2マス、正面1マスで、合計5箇所です。

続いて、向いている方向に応じてどの位置の壁を調べるかという情報を配列として定義します。  
wall\_front変数定義の下に、次のプログラムを追加してください。

int wall\_front[1] = {};  
**+**const int wall\_check\_list[4][5][2] = {  
**+**{ { 0, -1 },{ -1, -1 },{ 0, 1 },{ -1, 1 },{ -1, 0 } },  
**+**{ { -1, 0 },{ -1, 1 },{ 1, 0 },{ 1, 1 },{ 0, 1 } },  
**+**{ { 0, 1 },{ 1, 1 },{ 0, -1 },{ 1, -1 },{ 1, 0 } },  
**+**{ { 1, 0 },{ 1, -1 },{ -1, 0 },{ -1, -1 },{ 0, -1 } },  
**+**};

wall\_check\_list(うぉーる・ちぇっく・りすと)という変数は3重の配列、つまり3次元配列になっていますので、すこしややこしいです。しかし、それぞれの次元の大きさの意味が分かれば、実はそれほど難しいものではありません。  
最初の4という数値は、プレイヤーの向きの数を示しています。プレイヤーの向きは4方向あるので4というわけです。つまり、この変数は「プレイヤーの向きごとに異なるデータを持っている」ということになります。  
次の5という数値は調べるべきマス目の数です。つまり「調べるマスの位置ごとに異なるデータを持っている」ということです。  
最後の2という数値はダンジョンマップの次元数です。今回のダンジョンのデータはX軸とY軸の2次元の構造になっていますから「座標の軸ごとに異なるデータを持っている」必要があるわけです。  
これらをまとめると、wall\_check\_listは「プレイヤーの向き、調べるマスの位置、座標軸ごとに異なるデータを持っている配列」ということになります。

最後に、wall\_check\_listを使って各マスが壁かどうかを調べるプログラムを作ります。  
wall\_check\_list変数定義の下に、次のプログラムを追加してください。

{ { 1, 0 },{ 1, -1 },{ -1, 0 },{ -1, -1 },{ 0, -1 } },  
 };  
**+**wall\_left[0] = dungeon\_map  
**+** [player\_y + wall\_check\_list[player\_direction][0][0]]  
**+** [player\_x + wall\_check\_list[player\_direction][0][1]];  
**+**wall\_left[1] = dungeon\_map  
**+** [player\_y + wall\_check\_list[player\_direction][1][0]]  
**+** [player\_x + wall\_check\_list[player\_direction][1][1]];  
**+**wall\_right[0] = dungeon\_map  
**+** [player\_y + wall\_check\_list[player\_direction][2][0]]  
**+** [player\_x + wall\_check\_list[player\_direction][2][1]];  
**+**wall\_right[1] = dungeon\_map  
**+** [player\_y + wall\_check\_list[player\_direction][3][0]]  
**+** [player\_x + wall\_check\_list[player\_direction][3][1]];  
**+**wall\_front[0] = dungeon\_map  
**+** [player\_y + wall\_check\_list[player\_direction][4][0]]  
**+** [player\_x + wall\_check\_list[player\_direction][4][1]];

ちょっと長いですが、実際には3行単位でちょっとずつ違うプログラムが5セットあるだけです。3行書いたら残り4セットは「コピー&ペースト」して細部を修正する、といった技を駆使するといいでしょう。

**[NOTE]**  
コピー(マウス・トラックパッドの場合):  
マウスの左ドラッグ(左ボタンを押しながらマウスを動かして左ボタンを離す)して範囲を選択。  
範囲内で右クリックしてメニューを開き「コピー」を選ぶ。

コピー(キーボードの場合):  
shiftキーを押しながらカーソルキーを動かして範囲を選択し、shiftキーを離す。  
Ctrlキーを押しながらCキーを押す。

ペースト(マウス・トラックパッドの場合):  
ペーストしたい位置で右クリックしてメニューを開き「貼り付け」を選ぶ。

ペースト(キーボードの場合):  
ペーストしたい位置までカーソルキーでカーソル(点滅する縦棒)を移動する。  
Ctrlキーを押しながらVキーを押す。

もし配列を使わなかった場合、if文でプレイヤーの向きを調べて、向きごとに異なるプログラムを書かなければなりません。しかし、配列を使うことで上記のプログラムのようにひとつにまとめることができます。

## 壁を描く

ここまでで、向きを基準にした壁の位置は分かりました。この情報を使ってダンジョンを描いていきます。上記の壁の有無を調べるプログラムの下に、次のプログラムを追加してください。

wall\_front[0] = dungeon\_map  
[player\_y + wall\_check\_list[player\_direction][4][0]]  
[player\_x + wall\_check\_list[player\_direction][4][1]];  
**+**// 壁のある部分に画像を配置.  
**+**set\_image(0, 0, 0, "dungeon\_background.png");  
**+**if (wall\_left[1]) {  
**+** set\_image(1, 0, 0, "dungeon\_left\_wall\_1.png");  
**+**}  
**+**if (wall\_left[0]) {  
**+** set\_image(2, 0, 0, "dungeon\_left\_wall\_0.png");  
**+**}  
**+**if (wall\_right[1]) {  
**+** set\_image(3, 0, 0, "dungeon\_right\_wall\_1.png");  
**+**}  
**+**if (wall\_right[0]) {  
**+** set\_image(4, 0, 0, "dungeon\_right\_wall\_0.png");  
**+**}  
**+**if (wall\_front[0]) {  
**+** set\_image(5, 0, 0, "dungeon\_front\_wall\_0.png");  
**+**}

やっていることは「調べたマスに壁があるなら対応する画像を表示する」という作業を5マス分繰り返しているだけです。ここでもコピー&ペーストを上手く使ってください。  
なお、大きい管理番号を持つ画像ほど手前に表示されますので、プレイヤーに遠い壁ほど小さい管理番号を付けるようにしてあります。画像ファイルはshareフォルダにあります。

ここまで書けたら一度実行してみてください。  
3Dダンジョンの風景が表示されるでしょうか。

# ダンジョン内を移動する

## 行動選択

無事にダンジョンの風景を見ることができたら、次はダンジョン内を移動できるようにしてみましょう。  
とりあえず、プレイヤーは前進するか向きを変えるかを選択できることにしましょうか。  
ダンジョンの画像を表示するプログラムの下に、次のプログラムを追加してください。

if (wall\_front[0]) {  
 set\_image(5, 0, 0, "dungeon\_front\_wall\_0.png");  
 }  
**+  
+**// 行動を選択する.  
**+**set\_text(-360, 0, "どうしますか？");  
**+**const int player\_action =  
**+** select(-360, -40, 4, "前進", "右を向く", "左を向く", "後ろを向く");

行動の選択にはselect関数を使います。今回は、前進、右を向く、左を向く、後ろを向く、の4つのコマンドから選んでもらうようにしました。他にもRPGなんかだと「キャンプ」とか「道具」みたいなコマンドが考えられますが、そういった機能の追加はみなさんにお任せします。

## 前に進む

次に、それぞれの行動に対応するプログラムを書きます。上記の行動選択プログラムの下に、次のプログラムを追加してください。

const int player\_action =  
 select(-360, -40, 4, "前進", "右を向く", "左を向く", "後ろを向く");  
**+**if (player\_action == 0) {  
**+** if (wall\_front[0]) {  
**+** // 前方に壁がある場合は進めない.  
**+** reset\_text\_area(-400, -200, 800, 301);  
**+** set\_text(-360, 0, "壁があって進めない");  
**+** wait(1.5);  
**+** } else {  
**+** // 壁がなければ、プレイヤーの前方へ移動する.  
**+** const int move[4][2] = {{ 0, -1 },{ 1, 0 },{ 0, 1 },{ -1, 0 }};  
**+** player\_x += move[player\_direction][0];  
**+** player\_y += move[player\_direction][1];  
**+** }  
**+**} else {  
**+** // プレイヤーの向きを変える.  
**+** const int rotation\_count[4] = { 0, 1, 3, 2 };  
**+** player\_direction += rotation\_count[player\_action];  
**+** if (player\_direction >= 4) {  
**+** player\_direction -= 4;  
**+** }  
**+**}

player\_actionが0、つまり「前進」が選択された場合は、まず前方に壁があるかどうかを調べます。  
前方が壁かどうかはwall\_front[0]を見れば分かりますよね。この変数が1だったら壁があります。  
壁の場合は「進めない」というメッセージを表示するだけです。壁がなければプレイヤーの位置を変更しますが、もちろんプレイヤーの向きによって進む方向は異なります。ここは再び配列の出番です。  
向きは0～3の4つ、軸はXとYの2つなので「プレイヤーの向き」と「座標軸」ごとに異なるデータを持つ配列を作り、向きごとにXとYを増減させる数値を設定しておきます。  
例えば右向きは数値でいうと1です。また、右へ進むということは、X軸を1増やし、Y軸は変化しないということに相当します。そこで、move[1][0]には1を、move[1][1]には0を設定しているというわけです。  
あとはプログラムに書いてあるとおり、方向に対応するX座標とY座標の変化量を、プレイヤーの現在地をあらわす変数に足してやるだけです。

## 向きを変える

4つの行動のうち3つは「向きを変える」という同じ種類の行動なので、まとめてしまうことができます。  
向きは0～3の番号であらわされるのでした。また、向きには時計回りに番号が振られています。ということは、現在の向きに1を足すと、時計回りに90度向きを変えたことになるはずですよね。同様に2を足せば180度、3を足せば270度向きを変えられるはずです。そこで、コマンドごとに足すべき数値を決め、配列を作ります。あとはプログラムに書いてあるように、コマンドに対応する数値を配列から取り出して現在の向きに足せば、コマンドどおりに向きが変わります。  
ただし、向きとして有効なのは0～3の数値だけです。そこで、4以上になった場合は4を引いて0～3になるように修正しています。4というのは360度に相当しますが、360度というのは結局0度と同じ角度ですから、360度を引いても向きは同じなわけです。

ここまで書けたら実行してみてください。何かするたびにじゃんけんが始まるものの、とりあえず移動できるようになっていると思います。

## ゴール判定

おっと、ゴール地点に到着した場合のプログラムを忘れていました。ゴール判定がないと、永遠にダンジョンをさまよう羽目になってしまいます。プレイヤーの位置がゴール地点と等しければ、ゲームクリアということにしましょう。

ダンジョンを表示するプログラムと移動するプログラムの間に、次のプログラムを追加してください。

if (wall\_front[0]) {  
 set\_image(5, 0, 0, "dungeon\_front\_wall\_0.png");  
 }  
  
**+**if (player\_x == goal\_x && player\_y == goal\_y) {  
**+** // ゴールに到達したのでメッセージを表示してタイトルに戻る.  
**+** set\_text(-360, 0, "出口だ！");  
**+** set\_text(-360, -40, "おめでとう！");  
**+** set\_text(-360, -80, "あなたはダンジョンからの脱出に成功した！");  
**+** set\_text(-360, -120, "ＧＡＭＥ　ＣＬＥＡＲ");  
**+** wait(1);  
**+** set\_text(-360, -160, "(何かキーを押すとタイトルに戻ります)");  
**+** title\_flag = true;  
**+** wait\_any\_key();  
**+** wait(1);  
**+**} else {  
 // 行動を選択する.  
 set\_text(-360, 0, "どうしますか？");  
 const int player\_action =  
 select(-360, -40, 4, "前進", "右を向く", "左を向く", "後ろを向く");  
 if (player\_action == 0) {  
 if (wall\_front[0]) {  
 // 前方に壁がある場合は進めない.  
 reset\_text\_area(-400, -200, 800, 301);  
 set\_text(-360, 0, "壁があって進めない");  
 wait(1.5);  
 } else {  
 // 壁がなければ、プレイヤーの前方へ移動する.  
 const int move[4][2] = { { 0, -1 },{ 1, 0 },{ 0, 1 },{ -1, 0 } };  
 player\_x += move[player\_direction][0];  
 player\_y += move[player\_direction][1];  
 encount\_check\_flag = true;  
 }  
 } else {  
 // プレイヤーの向きを変える.  
 const int rotation\_count[] = { 0, 1, 3, 2 };  
 player\_direction += rotation\_count[player\_action];  
 if (player\_direction >= 4) {  
 player\_direction -= 4;  
 }  
 }  
**+**}

先ほど作成した移動プログラムは「ゴールしていなければ」という部分にあたりますので、最後に「}」を追加するのを忘れないように気をつけてください。

## じゃんけんを抑制する

移動のたびにじゃんけんしていてはダンジョンの探索がはかどりませんね。そこで、普段はじゃんけんをしないようにしておきます。  
まず、じゃんけん中かどうかをあらわす変数を追加します。ゴール位置をあらわす変数定義のしたに、次のプログラムを追加してください。

// ゴールの位置.  
 const int goal\_x = 3;  
 const int goal\_y = 1;  
  
**+**// 戦闘中はtrue. そうでなければfalse.  
**+**bool battle\_flag;

次にこの変数を初期化します。タイトル画面の最後に、次のプログラムを追加してください。

// ゲームの初期設定を行う.  
 player\_x = start\_x;  
 player\_y = start\_y;  
 player\_direction = dir\_down;  
 aiko\_flag = false;  
**+** battle\_flag = false;  
 }

それでは、この変数を使ってじゃんけんが始まらないようにしていきましょう。  
まずは、タイトル画面のプログラムの最後の閉じ波カッコ「}」にelse if文を付け足します。

aiko\_flag = false;  
 battle\_flag = false;  
 }  
**!**} else if (!battle\_flag) {  
 /\*  
 \* ダンジョン探索.  
 \*/  
 reset\_all\_image();  
 reset\_all\_text();

次に、ダンジョンの移動とじゃんけんの間にelse文を付け足します。

// プレイヤーの向きを変える.  
 const int rotation\_count[] = { 0, 1, 3, 2 };  
 player\_direction += rotation\_count[player\_action];  
 if (player\_direction >= 4) {  
 player\_direction -= 4;  
 }  
 }  
 }  
**+**} else {  
**+** /\*  
**+** \* じゃんけんバトル.  
**+** \*/  
 set\_text(-360, 260, "あなた(%d勝)", player\_win\_count);  
 set\_text(40, 260, "CPU(%d勝)", cpu\_win\_count);

最後に、じゃんけんの終わりに閉じ波括弧を付け足します。

reset\_all\_image(); // 次の勝負に備えてすべての画像を消す.  
 reset\_all\_text(); // 次の勝負に備えてすべての文字を消す.  
**+** }  
 }

括弧が増えてきたので、括弧を付け足す位置に気をつけてください。

ここまで書けたら実行してみてください。  
じゃんけんに煩わされずにダンジョンを探索できるようになっているでしょうか？

[課題 1]  
set\_text関数を使って、プレイヤーの位置と向きを表示してください。

# じゃんけんの復活

## じゃんけんの発生率

battle\_flag変数を導入したので、じゃんけんなしでダンジョンを探索できるようになりました。しかし、このままではせっかくのじゃんけんプログラムが全く動作しません。  
そこで、探索中に一定の確率でじゃんけんが始まるようにしてみましょう。  
まず、じゃんけんが発生する確率を変数として定義します。battle\_flag変数定義の下に、次のプログラムを追加してください。

　// 戦闘中のときはtrue. そうでなければfalse.  
　bool battle\_flag;  
**+**  
**+**// 敵との遭遇確率.  
**+**int encount\_percent = 10;  
　  
　/\*\*  
　\* アプリケーションの本体.  
　\*/  
　void application()

encount\_percent(えんかうんと・ぱーせんと)は「じゃんけんの発生率」を百分率で表したものです(とりあえず10%としてみました)。

## じゃんけんの発生を調べる

さて、じゃんけんの発生をどこで判定するかですが、ダンジョンの画像を表示した直後がいいでしょう。  
ダンジョンの画像を表示するプログラムの下に、次のプログラムを追加してください。

if (wall\_front[0]) {  
 set\_image(5, 0, 0, "dungeon\_front\_wall\_0.png");  
 }  
**+**  
**+**// 確率で敵と遭遇.  
**+**if (random(0, 99) < encount\_percent) {  
**+** set\_text(-360, 0, "敵だ！");  
**+** wait(2);  
**+** battle\_flag = true;  
**!**} else if (player\_x == goal\_x && player\_y == goal\_y) {  
 // ゴールに到達したのでメッセージを表示してタイトルに戻る.  
 set\_text(-360, 0, "出口だ！");

百分率なので、random関数で0～99の範囲の乱数を取得し、それがencount\_percent未満だったらじゃんけん発生！とします。  
じゃんけんが発生したらbattle\_flagにtrueを設定します。これによって、再びapplication関数が実行されたときには、ダンジョン探索プログラムではなくじゃんけんプログラムが動作する、という仕組みになっています。  
また、else文を使ってゴール判定のif文と接続することで、じゃんけんが発生した時はゴール判定と移動プログラムが実行されないようにしています。

書けたら実行してみてください。  
そして、ダンジョンを動き回ってじゃんけんが発生するか確かめてみましょう。

## じゃんけんの終了

さて、じゃんけんが発生してbattle\_flagがtrueになればじゃんけんが始まります。しかし、現在は一度じゃんけんが始まったらもう二度とダンジョン探索には戻れません。なぜならbattle\_flagがtrueだからです。

ダンジョン探索を続けられないというのはあまり面白くありません。そこで、じゃんけんに勝ったらダンジョン探索に戻れるようにしてみましょう。  
じゃんけんでプレイヤーが勝ったときのプログラムに、次のプログラムを追加してください。

play\_sound("correct4.mp3");  
 set\_image(4, 0, -150, "janken\_kachi.png");  
 player\_win\_count += 1; // プレイヤーの勝利数を1増やす.  
 aiko\_flag = false;  
**+**battle\_flag = false;

battle\_flagがtrueだからじゃんけんが続くのですから、じゃんけんを止めたかったらbattle\_flagにfalseを設定すればいいわけですね。

## じゃんけん画像の管理番号を変更する

現在、ダンジョンの画像とじゃんけんの画像は同じ管理番号を使っています。そのため、じゃんけんが始まるとダンジョンの風景がおかしくなってしまいます。そこで、ダンジョン画像の管理番号とじゃんけん画像の管理番号を分けようと思います。また、じゃんけんの画像はダンジョンの画像より手前に表示したいです。管理番号が大きい画像ほど手前に表示されるので、じゃんけん画像にはダンジョン画像より大きい管理番号を割り当てることにします。さらに、ダンジョンに表示するものが増えても大丈夫なように、じゃんけん画像の管理番号はダンジョン画像の管理番号からある程度離れた数値を使うようにします。  
これらの条件を満たし、なおかつ変更が難しくならないように、じゃんけん画像の新しい管理番号は「現在の管理番号に100を足した数値」としましょうか。

[課題 2]  
じゃんけんで使用している画像の管理番号を、100を足した新しい管理番号で置き換えてください。  
例:  
 set\_image(1, 0, 400, "janken\_aikode.png");  
 ↓  
set\_image(101, 0, 400, "janken\_aikode.png");  
 ^^^

## ダンジョンの風景を残す

さて、実はまだひとつやることが残っています。  
現在のじゃんけんは、最後にすべての文字と画像を消しています。このせいで、あいこだったり負けたりしたときはダンジョン画像が消えてしまいます。そこでじゃんけんの画像だけを消すように変更します。  
じゃんけんプログラムの最後の部分を次のように変更してください。

**-**reset\_all\_image(); // 次の勝負に備えてすべての画像を消す.  
**+**for (int i = 100; i < 200; ++i) {  
**+** reset\_image(i);  
**+**}  
 reset\_all\_text(); // 次の勝負に備えてすべての文字を消す.

今回は新しい変更記号として「-(マイナス)」が登場しています。これはプログラムから削除する行を示しています(プラスは追加する行、マイナスは削除する行ということです)。

reset\_all\_image関数の代わりにfor文とreset\_image関数を使って、じゃんけんの画像だけを消すようにしています(じゃんけん画像の管理番号は100～199までという想定です)。

書けたら実行してみてください。  
ダンジョンの風景が残ったままじゃんけんができたら成功です。

[課題 3]  
じゃんけん中、set\_image関数を使って対戦相手の画像を表示してください。  
対戦相手の画像を探すのが面倒な場合はshareフォルダのgoblin.pngを使うといいでしょう。

[課題 4]  
1. プレイヤーの体力をあらわす変数を定義し、じゃんけんに負けたら1ずつ減らすようにしてください。  
2. 体力が0以下になったらゲームオーバーと判定し、タイトル画面に戻るプログラムを追加してください。

[課題 5]  
1. 現在は、勝っても負けても「勝負を続けますか？」という選択肢が表示されます。  
 これを、ゲームオーバーになったときだけ表示するように変更してください。  
2. 表示する文字を「タイトル画面に戻りますか？」、選択肢を「戻る」「ゲームをやめる」に変更してください。  
 「戻る」が選ばれた時はタイトル画面に戻るようにしてください。

[課題 6]  
ゲームが始まった瞬間にじゃんけんが発生することがあります。  
encount\_percent変数の値を変更して、ゲームが始まった瞬間はじゃんけんが発生しないようにしてください。