ソフトゼミＢ　第３回

弾の作成

✤はじめに

前回のゼミでは自機と敵機が移動するだけのゲームを作りました。今回は弾を発射できるようにするのが目的です。ただし、あたり判定はまだ実装しませんので、本当に弾が出るだけです。もし余裕があるようなら、ソースコードを見ていって自分で進められるだけ進んでも構いません。今回も頑張っていきましょう。

✤前回の回答

前回穴を１３個埋めたと思います。こちらで想定している答えを書いていきます。ぶっちゃけ動けば多少違っても構わないので、参考程度に見ておいてください。

- 穴 2-1　　x
- 穴 2-2　　y
- 穴 2-3　　ENEMY
- 穴 2-4　　BULLET
- 穴 2-5　　-= player.vx

- 穴 2-6　　+= player.vy
- 穴 2-7　　-= player.vy
- 穴 2-8　　0
- 穴 2-9　　HEIGHT - player.size_y
- 穴 2-10　　enemy[i].vx

- 穴 2-11　　enemy[i].vy
- 穴 2-12　　enemy[i].hp == 0
- 穴 2-13　　t % 60 == 0

さて、今回もサンプルを使ってゲームを作っていきます。あらかじめ /* */ でコメントアウトしてある部分の一部を、コメントアウトを外して実装します。基本的には 穴 4 が書かれていないところをすべて解放していけば大丈夫ですが、こちらでどの部分を開放するか説明を兼ねて列挙していきますので確認してみてください。

移動に関する関数 (move.cpp)

void move_playerbuller()

void move_enembullet()

※ 今回は　move_item　関数は開放しません。

生成に関する関数(create.cpp)

void create_playerbullet()

void create_enemybullet()

void create_item(int x, int y, int size_x, int size_y)

描画に関する関数(draw.cpp)

各種 DrawGraph 関数すべて

これらを、「 // move_playerbuller() 」等といった記述が update.cpp にあるので、今回コメントアウトする分の関数と、それとともにプログラムの /**/ で囲まれたコメントアウトを外してください。場所は上記の () で囲ってある場所です。その上で、穴 3 を埋めたら今回の分は終了です。恐らく早く終わる人もいると思いますが、進められる人はガンガン進んでゲームを作っていってください。もし次回やる穴４も埋まったら、自主的にゲーム制作していってみてください。もちろん、何かわからなければ先輩にじゃんじゃん質問してください。

✤ 今回のソースコードの解説

さて、今回作るゲームはゲームステージ上で

敵を生成し、敵は弾を出す。(create_enemy、enemybullet)
自機は弾を Z キーで出し、(create_playerbullet)
自機は敵の弾が当たると HP が減る。(judge_enemy_to_player)
敵機も自機の出した弾が当たると HP が減る。(judge_player_to_enemy)

という流れがあります。今回では弾の生成と移動まで行いましたが、あたり判定、つまり judge 系関数を実装してないわけです。Judge 系関数については次回行うとして、今回はやや難しい create 系の関数を解説します。

```
//自機の弾の作成に関する関数
void create_playerbullet()
{
	static int t = 0;

	//Zキーが入力されていたら
	if (key[KEY_INPUT_Z])
	{
		//連続入力フレーム数を1増加
		t++;
		//3フレームごとに
		if (t % 3 == 0)
		{
			//すべての弾の中で
			for (int i = 0; i < BULLET; i++)
			{
				//存在していないものに対して
				if (playerbullet[i].hp == 0)
				{
```

```
                                        //存在を与える
                                        playerbullet[i].hp = 1;

                                        //座標を自機の正面に
playerbullet[i].x = player.x + (player.size_x - playerbullet[i].size_x) / 2.0;

playerbullet[i].y = (double)(player.y - playerbullet[i].size_y);

//1フレームでは1発だけ作りたいので、1発作ったらbreakで抜ける
                                        break;
                                    }
                                }
                            }
                        }
}
```

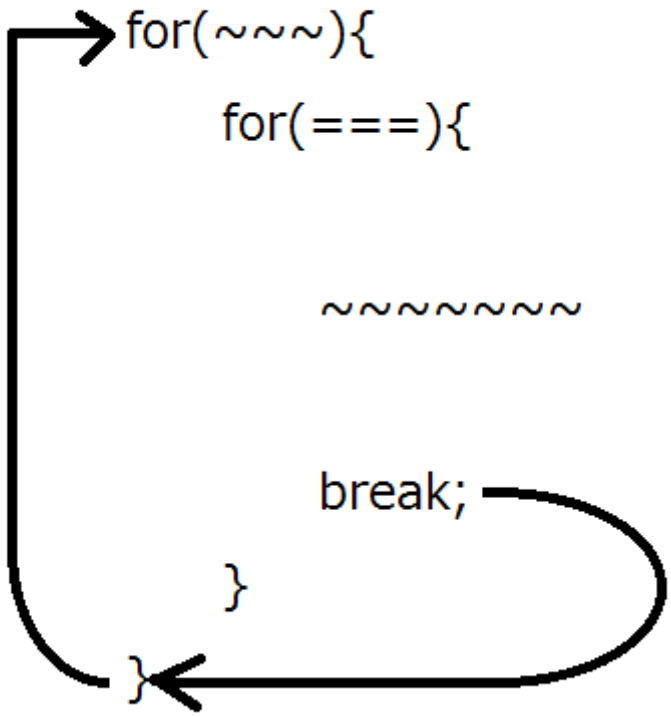
key[KEY_INPUT_Z]、t%3==0、break、弾の座標のあたりで注意が必要です。

key[KEY_INPUT_Z]について

DX ライブラリの関数の一つです。「Z キーの入力をチェックし、入力されれば 1 を返す」と、堅く言えますが、重要なのはこのほかのキーにも同じように使えることです。例えば、[KEY_INPUT_Z]を[KEY_INPUT_X]にすると X キーでも同じことができます。ＨＰの CheckHitKey 関数の説明を見れば、どのキーをどう対応させるかを確認できるのでぜひ一度見ておいてください。

t%3==0 は、「(自機の弾を) 三フレーム毎に (出す)」、という意味でそれ自体は問題なくわかると思います。しかしここで重要なのはこの 3 です。悪い意味で重要です。これはマジックナンバーというやつで、なぜ 3 なのかが、数字を見ただけではわかりません。プログラム全体の動きを把握して、t の挙動を理解しないと t%3==0 の意味が分からなくなってしまいます。その上、この値があちらこちらにあったとすると、複数の箇所で同じ値をいじる必要があります。これはまずいです。したがって、この 3 に、新たに変数を用意するなり define するなりして識別可能な文字列として表示するのが望ましいです。せっかくなので、余裕がある人だけで構いません。各自でこのマジックナンバーを消してみてください。

break の使用も注意が必要です。break はループ文を抜け出す効果がありますが、それで抜け出せるのは一つ分だけなので、二重以上のループでループを完全に抜けたいときは外側のループにも同様に break が必要です。つまり、一つのループに一つの break 文が必要ということです。



（外側の for 文がまだループする場合）

この挙動を利用することもできます。実際、create_enemybullet 関数ではこの挙動を利用しています。

```
//敵の弾の作成に関する関数
void create_enemybullet()
{
	//double ang;  //自機と敵の角度を格納する変数
	//すべての敵の中で
	for (int i = 0; i < ENEMY; i++)
	{
```

```
                //存在しているものに対して
                if (enemy[i].hp != 0)
                {
                        //その敵が作られてからのフレーム数を1増加させる
                        enemy[i].t++;
                        //その敵が作られてからenemy[i].freqフレームごとに
                        if (enemy[i].t % enemy[i].freq == 0)
                        {
                                //その敵の弾の中で
                                for (int j = 0; j < BULLET; j++)
                                {
                                        //存在していないものに対して
                                        if (enemybullet[i][j].hp == 0)
                                        {
                                                //存在を与える
                                        enemybullet[i][j].hp = 1;
                                                //座標を敵の正面に
                        enemybullet[i][j].x = enemy[i].x + enemy[i].size_x / 2;
                        enemybullet[i][j].y = enemy[i].y + enemy[i].size_y / 2;

                                                //自機狙いにしてみた
                                                //angに自機と敵との角度を入れる
//ang = atan2((double)((player.y + player.size_y / 2) - (enemy[i].y +
enemy[i].size_y)), (double)((player.x + player.size_x / 2) - (enemy[i].x +
enemy[i].size_x / 2)));
                                        //enemybullet[i][j].vx = cos(ang) * 5.0;
                                        //enemybullet[i][j].vy = sin(ang) * 5.0;

                                //1フレームでは1発だけ作りたいので、1発作ったらbreakで抜ける
                                                break;
                                        }
                                }
                        }
                }
        }
}
```

そのままだとややこしいですね。流れをざっくり説明します。

まず、HP!=0 な敵を探し、(外側の for 文)
その敵の弾を生成します。(内側の for 文)
生成する際、すでに存在している弾の枠を使わないように（ if(enemybullet.hp==0) ）
した上で敵の前方中央付近で（enemybullet.x 、y ）
弾を生成します。(enemybullet.hp=1)
もし生成できたらループを抜け（break)
別の敵においても同様のことをします。(外側の for 文)

ちなみにこの関数では enemy.freq という変数を用いて弾の生成頻度を管理しています。この freq という変数はどこから来たかというと、実は stage１.cpp にかく create_enemy 関数につながっています。このように、関数を用いて可変性を持たせたほうがより自由にゲーム制作ができます。

自機狙い弾は、atan2 でアークタンジェントを敵と自機の間で取って、そこで得た角度を使って弾を飛ばしています。興味があればコメントアウトを外すのもアリです。

それと、playerbullet の方と弾の生成する場所が違うことに気が付きましたか？違いは考えてみてください。特に答え合わせはしませんが、厳密なのは playerbullet の方です。

今回のゲーム制作はいわば、ゲームを作るためのプログラム、あるいはプログラムを書くための環境づくりです。一応動く、というものはもう皆さん作れているわけですが、もっとすごいものを作るためには、簡単に、様々な出力ができるようにすると良いです。そのためには関数が重要です。様々な動きができる関数を少しずつ作っていけば、秋に展示するゲームもより良いものになると思います。