```
#include "Block.h"
#include "DxLib.h"
#include "InputControl.h"

/*****************************************
* マクロ定義
*****************************************/
#define FIELD_HEIGHT       (21)             // フィールドのマスの高さ
#define FIELD_WIDTH              (12)              // フィールドのマスの幅
#define BLOCK_TROUT_SIZE   (4)                    // ブロックのマスサイズ
#define BLOCK_SIZE               (36)              // １ブロック当たりのサイズ
#define BLOCK_TYPE_MAX           (7)                       // 落ちてくるブロックの種類
#define BLOCK_NEXT_POS_X   (700)            // 次のブロックの座標(X座標)
#define BLOCK_NEXT_POS_Y   (500)            // 次のブロックの座標(Y座標)
#define BLOCK_STOCK_POS_X  (500)            // ストックされたブロックの座標(X座標)
#define BLOCK_STOCK_POS_Y  (350)            // ストックされたブロックの座標(Y座標)
#define DROP_BLOCK_INIT_X  (4)                    // 落ちてくるブロックの初期X座標
#define DROP_BLOCK_INIT_Y  (-1)             // 落ちてくるブロックの初期Y座標
#define DROP_SPEED               (60)              // 落下時間
#define TURN_CROCKWICE           (0)                       // 時計回りに回転させる
#define TURN_ANTICROCKWICE (1)                    // 反時計回りに回転させる

/*****************************************
* 型定義
*****************************************/
enum BLOCK_STATE
{
    E_BLOCK_EMPTY,                    // 空ブロック
    E_BLOCK_LIGHT_BLUE,               // 水色
    E_BLOCK_YELLOW_GREEN,         // 黄緑
    E_BLOCK_YELLOW,                   // 黄色
    E_BLOCK_ORANGE,                   // オレンジ
    E_BLOCK_BLUE,                     // 青
    E_BLOCK_PINK,                     // ピンク
    E_BLOCK_RED,                      // 赤
    E_BLOCK_GLAY,                     // 灰色
    E_BLOCK_WALL,                     // 壁
    E_BLOCK_IMAGE_MAX,
};

/*****************************************
* 定数定義
*****************************************/
const int C_BLOCK_TABLE[BLOCK_TYPE_MAX][BLOCK_TROUT_SIZE][BLOCK_TROUT_SIZE] = {
    {
          {0, 0, 0, 0},
          {0, 1, 1, 0},
```

```
        {0, 1, 1, 0},
        {0, 0, 0, 0}
    },
    {
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {2, 2, 2, 2},
        {0, 0, 0, 0}
    },
    {
        {0, 0, 0, 0},
        {3, 0, 0, 0},
        {3, 3, 3, 0},
        {0, 0, 0, 0}
    },
    {
        {0, 0, 0, 0},
        {0, 0, 0, 4},
        {0, 4, 4, 4},
        {0, 0, 0, 0}
    },
    {
        {0, 0, 0, 0},
        {0, 5, 5, 0},
        {0, 0, 5, 5},
        {0, 0, 0, 0}
    },
    {
        {0, 0, 0, 0},
        {0, 6, 6, 0},
        {6, 6, 0, 0},
        {0, 0, 0, 0}
    },
    {
        {0, 0, 0, 0},
        {0, 7, 0, 0},
        {7, 7, 7, 0},
        {0, 0, 0, 0}
    }
};

/*****************************************
* 変数宣言
*****************************************/
int BlockImage[E_BLOCK_IMAGE_MAX];                                  // ブロック画像
BLOCK_STATE Field[FIELD_HEIGHT][FIELD_WIDTH];                       // フィールド配列
BLOCK_STATE Next[BLOCK_TROUT_SIZE][BLOCK_TROUT_SIZE];               // 待機状態のブロック
```

```
BLOCK_STATE Stock[BLOCK_TROUT_SIZE][BLOCK_TROUT_SIZE];          // ストックのブロック
BLOCK_STATE DropBlock[BLOCK_TROUT_SIZE][BLOCK_TROUT_SIZE];      // 落ちるブロック
int DropBlock_X;                                                                    // 落ち
るブロックのX座標
int DropBlock_Y;                                                                    // 落ち
るブロックのY座標

int WaitTime;         // 待機時間
int Stock_Flg;        // ストックフラグ
int Generate_Flg;     // 生成フラグ
int DeleteLine;       // 消したラインの数
int SoundEffect[3];   // SE

/*****************************************
* プロトタイプ宣言
*****************************************/
void create_field(void);                // フィールドの生成処理
void create_block(void);                // ブロックの生成処理
void move_block(void);                      // ブロックの移動処理
void change_block(void);                // ストック交換処理
void turn_block(int clockwise);         // ブロック回転処理
int  check_overlap(int x, int y);// 範囲外チェック処理
void lock_block(int x, int y);          // 着地したブロックを固定済みに変更する処理
void check_line(void);                      // ブロックの横一列確認処理

/*****************************************
* ブロック機能：初期化処理
* 引　数：なし
* 戻り値：エラー情報（-1:異常，それ以外:正常）
*****************************************/
int Block_Initialize(void)
{
    int ret = 0;      // 戻り値
    int i = 0;

    // ブロック画像の読込み
    ret = LoadDivGraph("images/block.png", E_BLOCK_IMAGE_MAX, 10, 1, BLOCK_SIZE,
BLOCK_SIZE, BlockImage);

    // SEの読み込み
    SoundEffect[0] = LoadSoundMem("sounds/SE3.mp3");
    SoundEffect[1] = LoadSoundMem("sounds/SE4.mp3");
    SoundEffect[2] = LoadSoundMem("sounds/SE5.wav");

    // 音量の調整
    ChangeVolumeSoundMem(150, SoundEffect[0]);
    ChangeVolumeSoundMem(150, SoundEffect[1]);
```

```
    ChangeVolumeSoundMem(130, SoundEffect[2]);

    // フィールドの生成
    create_field();

    // ブロック生成
    create_block();
    create_block();

    // 待機時間の初期化
    WaitTime = 0;
    // ストックフラグの初期化
    Stock_Flg = FALSE;
    // 生成フラグの初期化
    Generate_Flg = TRUE;
    // 消したラインの数の初期化
    DeleteLine = 0;

    // エラーチェック
    for (i = 0; i < 3; i++)
    {
          if (SoundEffect[i] == -1)
          {
                ret = -1;
                break;
          }
    }

    return ret;
}

/*****************************************
* ブロック機能：更新処理
* 引　数：なし
* 戻り値：なし
*****************************************/
void Block_Update(void)
{
    // ブロックの移動処理
    move_block();

    // ブロックのストック
    if ((GetButtonDown(XINPUT_BUTTON_LEFT_SHOULDER) == TRUE) ||
          (GetButtonDown(XINPUT_BUTTON_RIGHT_SHOULDER) == TRUE))
    {
          // 生成可能であれば
          if (Generate_Flg == TRUE)
```

```
		{
			change_block();	// ストック交換処理
			// ブロックの回転を正位置にする
		}
	}

	// ブロックの回転（反時計回り）
	if ((GetButtonDown(XINPUT_BUTTON_A) == TRUE) ||
		(GetButtonDown(XINPUT_BUTTON_Y) == TRUE))
	{
		turn_block(TURN_ANTICROCKWICE);
	}
	// ブロックの回転（時計回り）
	if ((GetButtonDown(XINPUT_BUTTON_B) == TRUE) ||
		(GetButtonDown(XINPUT_BUTTON_X) == TRUE))
	{
		turn_block(TURN_CROCKWICE);
	}

	// 落下処理
	WaitTime++;		// カウンタの更新
	if (WaitTime > DROP_SPEED)
	{
		if (check_overlap(DropBlock_X, DropBlock_Y + 1) == TRUE)
		{
			DropBlock_Y++;
		}
		else
		{
			// ブロックの固定
			lock_block(DropBlock_X, DropBlock_Y);
			// ブロックの消去とブロックを下ろす処理
			check_line();
			// 新しいブロックの生成
			create_block();
		}
		// カウンタの初期化
		WaitTime = 0;
	}
}

/*****************************************
* ブロック機能：描画処理
* 引　数：なし
* 戻り値：なし
*****************************************/
void Block_Draw(void)
```

```
{
    int i, j;          // ループカウンタ

    // フィールドのブロックを描画
    for (i = 0; i < FIELD_HEIGHT; i++)
    {
          for (j = 0; j < FIELD_WIDTH; j++)
          {
                if (Field[i][j] != E_BLOCK_WALL)
                {
                      DrawGraph(j * BLOCK_SIZE, i * BLOCK_SIZE, BlockImage[Field[i][j]],
TRUE);
                }
          }
    }
    // 次のブロックとストックされたブロックを描画
    for (i = 0; i < BLOCK_TROUT_SIZE; i++)
    {
          for (j = 0; j < BLOCK_TROUT_SIZE; j++)
          {
                // 次のブロックを描画
                DrawGraph(BLOCK_SIZE * j + BLOCK_NEXT_POS_X, BLOCK_SIZE * i +
BLOCK_NEXT_POS_Y, BlockImage[Next[i][j]], TRUE);
                // ストックされたブロックを描画
                DrawGraph(BLOCK_SIZE * j + BLOCK_STOCK_POS_X, BLOCK_SIZE * i +
BLOCK_STOCK_POS_Y, BlockImage[Stock[i][j]], TRUE);
          }
    }
    // 落ちてくるブロックの描画
    for (i = 0; i < BLOCK_TROUT_SIZE; i++)
    {
          for (j = 0; j < BLOCK_TROUT_SIZE; j++)
          {
                DrawGraph((DropBlock_X + j) * BLOCK_SIZE, (DropBlock_Y + i) * BLOCK_SIZE,
BlockImage[DropBlock[i][j]], TRUE);
          }
    }

}

/*****************************************
* ブロック機能：ブロックの生成判定処理
* 引　数：なし
* 戻り値：TRUE(ブロックの生成ができる),FALSE(生成不可)
*****************************************/
int Get_GenerateFlg(void)
{
```

```
    return Generate_Flg;
}

/*****************************************
* ブロック機能：消したラインの数取得処理
* 引　数：なし
* 戻り値：消したラインの数
*****************************************/
int Get_Line(void)
{
    return DeleteLine;
}

/*****************************************
* ブロック機能：フィールド生成処理
* 引　数：なし
* 戻り値：なし
*****************************************/
void create_field(void)
{
    int i, j;         // ループカウンタ

    // フィールドの生成
    for (i = 0; i < FIELD_HEIGHT; i++)
    {
          for (j = 0; j < FIELD_WIDTH; j++)
          {

                // フィールド値の設定
                if (j == 0 || j == FIELD_WIDTH - 1 || i == FIELD_HEIGHT - 1)
                {
                      Field[i][j] = E_BLOCK_WALL;         // 壁状態にする
                }
                else
                {
                      Field[i][j] = E_BLOCK_EMPTY;  // 空状態にする
                }
          }
    }

}

/*****************************************
* ブロック機能：ブロック生成処理
* 引　数：なし
* 戻り値：なし
*****************************************/
```

```
void create_block(void)
{
    int i, j;          // ループカウンタ
    int block_type;    // 次に出現させるブロックタイプ

    // 次に出現させるブロックの決定する
    block_type = GetRand(BLOCK_TYPE_MAX - 1);

    // 新しいブロックをセット＆次のブロックを生成
    for (i = 0; i < BLOCK_TROUT_SIZE; i++)
    {
          for (j = 0; j < BLOCK_TROUT_SIZE; j++)
          {
                DropBlock[i][j] = Next[i][j];
                Next[i][j] = (BLOCK_STATE)C_BLOCK_TABLE[block_type][i][j];
          }
    }

    // 出現位置の設定
    DropBlock_X = DROP_BLOCK_INIT_X;
    DropBlock_Y = DROP_BLOCK_INIT_Y;

    // 生成できなかった時、ゲームオーバーに遷移する
    if (check_overlap(DropBlock_X, DropBlock_Y) == FALSE)
    {
          Generate_Flg = FALSE;
    }
}

/*****************************************
* ブロック機能：ブロックの移動処理
* 引　数：なし
* 戻り値：なし
*****************************************/
void move_block(void)
{
    // 左入力時
    if (GetButtonDown(XINPUT_BUTTON_DPAD_LEFT))
    {
          if (check_overlap(DropBlock_X - 1, DropBlock_Y) == TRUE)
          {
                DropBlock_X--;
          }

    }

    // 右入力時
```

```
    if (GetButtonDown(XINPUT_BUTTON_DPAD_RIGHT))
    {
          if (check_overlap(DropBlock_X + 1, DropBlock_Y) == TRUE)
          {
                DropBlock_X++;
          }
    }

    // 上入力時(ハードドロップ処理)
    if (GetButtonDown(XINPUT_BUTTON_DPAD_UP))
    {
          while (check_overlap(DropBlock_X, DropBlock_Y + 1) == TRUE)
          {
                DropBlock_Y++;
          }
    }

    // 下入力時(ソフトドロップ処理)
    if (GetButton(XINPUT_BUTTON_DPAD_DOWN))
    {
          if (check_overlap(DropBlock_X, DropBlock_Y + 1) == TRUE)
          {
                DropBlock_Y++;
          }
    }
}

/*****************************************
* ブロック機能：ストック交換処理
* 引　数：なし
* 戻り値：なし
*****************************************/
void change_block(void)
{
    BLOCK_STATE temp[BLOCK_TROUT_SIZE][BLOCK_TROUT_SIZE] = { E_BLOCK_EMPTY };     // 退避
領域
    int i, j;          // ループカウンタ

    // ストック先が空かどうか確認
    if (Stock_Flg == TRUE)
    {
          for (i = 0; i < BLOCK_TROUT_SIZE; i++)
          {
                for (j = 0; j < BLOCK_TROUT_SIZE; j++)
                {
                      temp[i][j] = DropBlock[i][j];
                      DropBlock[i][j] = Stock[i][j];
```

```
                        Stock[i][j] = temp[i][j];
                }
        }
    }
    else
    {
        Stock_Flg = TRUE;
        for (i = 0; i < BLOCK_TROUT_SIZE; i++)
        {
                for (j = 0; j < BLOCK_TROUT_SIZE; j++)
                {
                        Stock[i][j] = DropBlock[i][j];
                }
        }
        // 新しいブロックの設定と次のブロックの生成
        create_block();
    }

}

/*****************************************
* ブロック機能：ブロックの交換処理
* 引　数：回転指せる向き（0:時計回り 1:反時計回り）
* 戻り値：なし
*****************************************/
void turn_block(int clockwise)
{
    BLOCK_STATE temp[BLOCK_TROUT_SIZE][BLOCK_TROUT_SIZE] = { E_BLOCK_EMPTY };     // 退避
領域
    int i, j;         // ループカウンタ

    do
    {
        if (clockwise == TURN_CROCKWICE)
        {
                // ブロックを一時保持する
                for (i = 0; i < BLOCK_TROUT_SIZE; i++)
                {
                        for (j = 0; j < BLOCK_TROUT_SIZE; j++)
                        {
                                temp[j][3 - i] = DropBlock[i][j];
                        }
                }
        }
        else
        {
                // ブロックを一時保持する
```

```
			for (i = 0; i < BLOCK_TROUT_SIZE; i++)
			{
				for (j = 0; j < BLOCK_TROUT_SIZE; j++)
				{
					temp[3 - j][i] = DropBlock[i][j];
				}
			}
		}

		// ブロック回転
		for (i = 0; i < BLOCK_TROUT_SIZE; i++)
		{
			for (j = 0; j < BLOCK_TROUT_SIZE; j++)
			{
				DropBlock[i][j] = temp[i][j];
			}
		}

		// 壁側の補正処理
		if (check_overlap(DropBlock_X, DropBlock_Y) && DropBlock_X >= E_BLOCK_WALL)
		{
			DropBlock_X--;
		}
		if (check_overlap(DropBlock_X, DropBlock_Y) && DropBlock_X <= E_BLOCK_EMPTY)
		{
			DropBlock_X++;
		}

	} while (check_overlap(DropBlock_X, DropBlock_Y) == FALSE);

	PlaySoundMem(SoundEffect[2], DX_PLAYTYPE_BACK, TRUE);
}

/*****************************************
* ブロック機能：範囲外チェック処理
* 引　数：落下ブロックの座標（x, y）
* 戻り値：TRUE(範囲内), FALSE(範囲外)
*****************************************/
int check_overlap(int x, int y)
{
	int i, j;		// ループカウンタ

	for (i = 0; i < BLOCK_TROUT_SIZE; i++)
	{
		for (j = 0; j < BLOCK_TROUT_SIZE; j++)
		{
			if (DropBlock[i][j] != E_BLOCK_EMPTY)
```

```
			{
				if (Field[i + y][j + x] != E_BLOCK_EMPTY)
				{
					return FALSE;
				}
			}
		}
	}

	return TRUE;
}

/*****************************************
* ブロック機能：着地したブロックを固定済みにする処理
* 引　数：落下ブロックの座標（x, y）
* 戻り値：なし
*****************************************/
void lock_block(int x, int y)
{
	int i, j;		// ループカウンタ

	for (i = 0; i < BLOCK_TROUT_SIZE; i++)
	{
		for (j = 0; j < BLOCK_TROUT_SIZE; j++)
		{
			if (DropBlock[i][j] != E_BLOCK_EMPTY)
			{
				Field[y + i][x + j] = DropBlock[i][j];
			}
		}
	}
	PlaySoundMem(SoundEffect[1], DX_PLAYTYPE_BACK, TRUE);
}

/*****************************************
* ブロック機能：ブロックの横一列確認処理
* 引　数：なし
* 戻り値：なし
*****************************************/
void check_line(void)
{
	int i, j, k;			// ループカウンタ

	for (i = 0; i < FIELD_HEIGHT - 1; i++)
	{
		for (j = 1; j < FIELD_WIDTH; j++)
		{
```

```
			// 行の途中が空いているか？
			if (Field[i][j] == E_BLOCK_EMPTY)
			{
				break;
			}
		}

		// 一列揃っていたら、カウントを増やし、1段下げる
		if (j >= FIELD_WIDTH)
		{
			// カウントを増加
			DeleteLine++;

			// 1段下げる
			for (k = i; k > 0; k--)
			{
				for (j = 1; j < FIELD_WIDTH; j++)
				{
					Field[k][j] = Field[k - 1][j];
				}
			}
			PlaySoundMem(SoundEffect[0], DX_PLAYTYPE_BACK, TRUE);
		}
	}
}
```