```
#include "Stage.h"
#include "DxLib.h"
#include "InputControl.h"
/*******************************
* マクロ定義
*******************************/
#define HEIGHT                  (12)        // ブロック配置サイズ（高さ）
#define WIDTH                   (12)        // ブロック配置サイズ（幅）
#define BLOCKSIZE               (48)        // ブロックサイズ
#define BLOCK_IMAGE_MAX         (10)        // ブロック画像数

#define ITEM_MAX                (8)             // アイテム最大数

#define SELECT_CURSOR           (0)
#define NEXT_CURSOR             (1)
#define TMP_CURSOR              (2)
/*******************************
* 型定義
*******************************/
typedef struct
{
    int flg;
    int x, y;
    int width, height;
    int image;
    int backup;
}T_Object;
```

```
typedef struct
{
	int x;
	int y;
}T_CURSOR;

enum
{
	E_NONE,
	E_ONCE,
	E_SECOND
};

/*******************************
* 変数宣言
*******************************/
T_Object Block[HEIGHT][WIDTH];			// ブロックオブジェクトデータ
T_CURSOR Select[3];					// セレクトカーソル座標
int Item[ITEM_MAX];
int ClickStatus;
int Stage_State;
int Stage_Mission;
int Stage_Score;
int ClearFlag;

int BlockImage[BLOCK_IMAGE_MAX];		// ブロック画像
int StageImage;					// 背景用画像
```

```
int ClickSE;                                    // クリックSE
int FadeOutSE;                                  // フェードアウトSE
int MoveBlockSE;                                // ブロック移動SE

/*********************************
* プロトタイプ宣言
*********************************/
int combo_check(int x, int y);
void combo_check_h(int y, int x, int* cnt, int* col);
void combo_check_w(int y, int x, int* cnt, int* col);
void save_block(void);
void restore_block(void);

/*********************************
* ステージ制御機能：初期化処理
* 引　数：なし
* 戻り値：エラー情報
*********************************/
int StageInitialize(void)
{

	int ret = 0;
	int i;

	// 画像読み込み
	LoadDivGraph("images/block.png", BLOCK_IMAGE_MAX, BLOCK_IMAGE_MAX, 1,
BLOCKSIZE, BLOCKSIZE, BlockImage);
	StageImage = LoadGraph("images/stage.png");
```

```
// 音源読み込み
ClickSE = LoadSoundMem("sounds/click_se.mp3");
FadeOutSE = LoadSoundMem("sounds/fadeout_se.mp3");
MoveBlockSE = LoadSoundMem("sounds/moveblock_se.mp3");

// ブロック生成処理
CreateBlock();

ClickStatus = E_NONE;
Stage_State = 0;
Stage_Score = 0;
ClearFlag = FALSE;

for (i = 0; i < 3; i++)
{
	Select[i].x = 0;
	Select[i].y = 0;
}

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

```
	if (StageImage == -1)
	{
		ret = -1;
	}
	if (ClickSE == -1)
	{
		ret = -1;
	}
	if (FadeOutSE == -1)
	{
		ret = -1;
	}
	if (MoveBlockSE == -1)
	{
		ret = -1;
	}



	return ret;
}

/*******************************
* ステージ制御機能：ステージの描画
* 引　数：なし
* 戻り値：なし
*******************************/
void StageDraw(void) {
```

```
	DrawGraph(0, 0, StageImage, FALSE);

	//アイテムの取得個数を描画
	for (int i = 0; i < ITEM_MAX; i++)
	{
		DrawRotaGraph(540, 245 + i * 30, 0.5f, 0, BlockImage[i + 1], TRUE, 0);
		DrawFormatString(580, 235 + i * 30, 0xffffff, "%3d", Item[i]);
	}

	//ブロックを描画
	for (int i = 0; i < HEIGHT; i++)
	{
		for (int j = 0; j < WIDTH; j++)
		{
			if (Block[i][j].flg == TRUE && Block[i][j].image != NULL)
			{
				DrawGraph(Block[i][j].x, Block[i][j].y,
BlockImage[Block[i][j].image], TRUE);
			}
		}
	}

	//選択ブロックを描画
	DrawGraph(Select[SELECT_CURSOR].x * BLOCKSIZE, Select[SELECT_CURSOR].y *
BLOCKSIZE, BlockImage[9], TRUE);
	if (ClickStatus != E_NONE)
	{
```

```
		DrawGraph(Select[NEXT_CURSOR].x * BLOCKSIZE,
Select[NEXT_CURSOR].y * BLOCKSIZE, BlockImage[9], TRUE);
	}

	//ミッションを描画
	SetFontSize(20);
	DrawFormatString(590, 211, GetColor(255, 255, 255), "%3d", Stage_Mission);

	//アイテムの取得個数を描画
	for (int i = 0; i < ITEM_MAX; i++)
	{
		DrawRotaGraph(540, 245 + i * 30, 0.5f, 0, BlockImage[i + 1], TRUE, 0);
		DrawFormatString(580, 235 + i * 30, GetColor(255, 255, 255), "%3d",
Item[i]);
	}
}

/*******************************
* ステージ制御機能：ブロック生成処理
* 引　数：なし
* 戻り値：なし
*******************************/
void CreateBlock(void)
{
	int Check = 0;
	int i, j;

	do
```

```
{
	Check = 0;
	for (i = 0; i < HEIGHT; i++)
	{
		for (j = 0; j < WIDTH; j++)
		{
			if (j == 0 || j == WIDTH - 1 || i == HEIGHT - 1 || i == 0)
			{
				Block[i][j].flg = FALSE;
				Block[i][j].image = NULL;
			}
			else
			{
				Block[i][j].flg = TRUE;
				Block[i][j].x = (j - 1) * BLOCKSIZE;
				Block[i][j].y = (i - 1) * BLOCKSIZE;
				Block[i][j].width = BLOCKSIZE;
				Block[i][j].height = BLOCKSIZE;
				Block[i][j].image = GetRand(7) + 1; // 1～8の乱数
			}
		}
	}

	/*for (i = 1; i < HEIGHT - 1; i++)
	{
		for (j = 1; j < WIDTH - 1; j++)
		{
			if (Block[i][j].image == NULL)
```

```
                        {
                                Block[i][j].image = GetRand(7) + 1;
                        }
                }
        }*/
        //ブロック連鎖チェック
        for (i = 1; i < HEIGHT - 1; i++)
        {
                for ( j = 1; j < WIDTH - 1; j++)
                {
                        Check += combo_check(i, j);
                }
        }
    } while (Check != 0);

    for (i = 0; i < ITEM_MAX; i++)
    {
        Item[i] = 0;
    }
}

/*******************************
* ステージ制御機能：ブロック選択処理
* 引　数：なし
* 戻り値：なし
*******************************/
void SelectBlock(void)
{
```

```
int TmpBlock;
int Result;

// カーソル座標の取得
Select[SELECT_CURSOR].x = GetMousePositionX() / BLOCKSIZE;
Select[SELECT_CURSOR].y = GetMousePositionY() / BLOCKSIZE;

//選択ブロックの範囲を制御
if (Select[SELECT_CURSOR].x < 0)
{
	Select[SELECT_CURSOR].x = 0;
}
if (Select[SELECT_CURSOR].x > WIDTH - 3)
{
	Select[SELECT_CURSOR].x = WIDTH - 3;
}
if (Select[SELECT_CURSOR].y < 0)
{
	Select[SELECT_CURSOR].y = 0;
}
if (Select[SELECT_CURSOR].y > HEIGHT - 3)
{
	Select[SELECT_CURSOR].y = HEIGHT - 3;
}

//クリックでブロックを選択
if (GetKeyFlg(MOUSE_INPUT_LEFT)) {
```

```
		//クリック効果音
		PlaySoundMem(ClickSE, DX_PLAYTYPE_BACK);

		if (ClickStatus == E_NONE) {
			Select[NEXT_CURSOR].x = Select[SELECT_CURSOR].x;
			Select[NEXT_CURSOR].y = Select[SELECT_CURSOR].y;
			ClickStatus = E_ONCE;
		}
		else if (ClickStatus == E_ONCE &&
				((abs(Select[NEXT_CURSOR].x - Select[SELECT_CURSOR].x)
== 1 &&
				(abs(Select[NEXT_CURSOR].y - Select[SELECT_CURSOR].y)
== 0)) ||
				(abs(Select[NEXT_CURSOR].x - Select[SELECT_CURSOR].x)
== 0 &&
				abs(Select[NEXT_CURSOR].y - Select[SELECT_CURSOR].y) ==
1)))
		{

			Select[TMP_CURSOR].x = Select[SELECT_CURSOR].x;
			Select[TMP_CURSOR].y = Select[SELECT_CURSOR].y;
			ClickStatus = E_SECOND;
		}
	}


	//選択ブロックを交換する。
	if (ClickStatus == E_SECOND)
```

```
    {
        TmpBlock = Block[Select[NEXT_CURSOR].y + 1][Select[NEXT_CURSOR].x +
1].image;
        Block[Select[NEXT_CURSOR].y + 1][Select[NEXT_CURSOR].x + 1].image =
Block[Select[TMP_CURSOR].y + 1][Select[TMP_CURSOR].x + 1].image;
        Block[Select[TMP_CURSOR].y + 1][Select[TMP_CURSOR].x + 1].image =
TmpBlock;

        //連鎖が3つ以上か調べる。
        Result = 0;
        Result += combo_check(Select[NEXT_CURSOR].y + 1,
Select[NEXT_CURSOR].x + 1);
        Result += combo_check(Select[TMP_CURSOR].y + 1,
Select[TMP_CURSOR].x + 1);

        //連鎖が3未満なら選択ブロックを元に戻す
        if (Result == 0)
        {

            int TmpBlock = Block[Select[NEXT_CURSOR].y +
1][Select[NEXT_CURSOR].x + 1].image;
            Block[Select[NEXT_CURSOR].y + 1][Select[NEXT_CURSOR].x +
1].image = Block[Select[TMP_CURSOR].y + 1][Select[TMP_CURSOR].x + 1].image;
            Block[Select[TMP_CURSOR].y + 1][Select[TMP_CURSOR].x +
1].image = TmpBlock;
        }
        else
        {
```

```
                //連鎖が3つ以上ならブロックを消しブロック移動処理へ移行する
                Stage_State = 1;
            }

            //次にクリックできるようにClockFlagを0にする
            ClickStatus = E_NONE;
        }
}

/*******************************
* ステージ制御機能：フェードアウト処理
* 引　数：なし
* 戻り値：なし
*******************************/
void FadeOutBlock(void)
{
    static int BlendMode = 255;
    int i, j;

    //フェードアウト効果音
    if (CheckSoundMem(FadeOutSE) == 0)
    {
        PlaySoundMem(FadeOutSE, DX_PLAYTYPE_BACK);
    }

    //描画モードをアルファブレンドにする
    SetDrawBlendMode(DX_BLENDGRAPHTYPE_ALPHA, BlendMode);
    for (i = 1; i < HEIGHT - 1; i++)
```

```
	{
		for (j = 1; j < WIDTH - 1; j++)
		{
			if (Block[i][j].image == 0)
			{
				DrawGraph(Block[i][j].x, Block[i][j].y,
BlockImage[Block[i][j].backup], TRUE);
			}
		}
	}

	//描画モードをノーブレンドにする
	SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 0);

	BlendMode -= 5;

	if (BlendMode == 0)
	{
		BlendMode = 255;
		Stage_State = 2;
		StopSoundMem(FadeOutSE);
	}
}

/*******************************
* ステージ制御機能：ブロック移動処理
* 引　数：なし
* 戻り値：なし
```

```
*******************************/
void MoveBlock(void)
{
	int i, j, k;
	// ブロック移動効果音
	PlaySoundMem(MoveBlockSE, DX_PLAYTYPE_BACK);

	// ↓へ移動する処理
	for (i = 1; i < HEIGHT - 1; i++)
	{
		for (j = 1; j < WIDTH - 1; j++)
		{
			if (Block[i][j].image == 0)
			{
				for (k = i; k > 0; k--)
				{
					Block[k][j].image = Block[k - 1][j].image;
					Block[k - 1][j].image = 0;
				}
			}
		}
	}

	// 空のブロックを生成する処理
	for (i = 1; i < HEIGHT - 1; i++)
	{
		for (j = 1; j < WIDTH - 1; j++)
		{
```

```
                if (Block[i][j].image == 0)
                {
                        Block[i][j].image = GetRand(7) + 1;
                }
        }
    }

    // 連鎖チェックへ移行する
    Stage_State = 3;
}

/*******************************
* ステージ制御機能：連鎖チェック処理
* 引　数：なし
* 戻り値：なし
*******************************/
void CheckBlock(void)
{
    int Result = 0;
    int i, j;

    //ブロック連鎖チェック
    for (i = 1; i < HEIGHT - 1; i++)
    {
        for (j = 1; j < WIDTH - 1; j++)
        {
            Result += combo_check(i, j);
        }
```

```
	}

	//連鎖がなくなればブロック選択へ
	//そうでなければブロック移動へ移行して連鎖チェックを継続する
	if (Result == 0)
	{
		//クリアチェック処理へ移行する。
		Stage_State = 4;
	}
	else
	{
		//連鎖が3つ以上ならブロックを消しブロック移動処理へ移行する
		Stage_State = 1;
	}
}

/*******************************
* ステージ制御機能：クリア条件チェック処理
* 引　数：なし
* 戻り値：なし
* 備　考：クリア条件フラグを0とし、各スクールの削除ブロックが
*　　　　レベルよりも数が少なかったらチェック処理を中断してゲームを続行する。
*******************************/
void CheckClear(void)
{

	int i;
```

```
	for (i = 0; i < ITEM_MAX; i++)
	{
		if (Item[i] >= Stage_Mission)
		{
			ClearFlag = TRUE;
			break;
		}
	}
	if (ClearFlag != TRUE)
	{
		Stage_State = 0;
	}
}

/*******************************
* ステージ制御機能：ステージステータス情報取得処理
* 引　数：なし
* 戻り値：ステージのステータス情報
*******************************/
int Get_StageState(void)
{
	return Stage_State;
}

/*******************************
* ステージ制御機能：ミッション情報取得処理
* 引　数：なし
* 戻り値：ミッションがクリアしているか
```

```
*********************************/
int Get_StageClearFlag(void)
{
	return ClearFlag;
}

/*********************************
* ステージ制御機能：ミッション情報取得処理
* 引　数：なし
* 戻り値：ミッションがクリアしているか
*********************************/
int Get_StageScore(void)
{
	return Stage_Score;
}

/*********************************
* ステージ制御機能：ミッション情報取得処理
* 引　数：次ミッションに必要な数値
* 戻り値：なし
*********************************/
void Set_StageMission(int mission)
{
	Stage_Mission += mission;
}

/*********************************
* ステージ制御機能：連鎖チェック処理
```

```
* 引数１：ブロックYマス
* 引数２：ブロックXマス
* 戻り値：連鎖有無（0:無し　1:有り）
*********************************/
int combo_check(int y, int x)
{
	int ret = FALSE;

	// 縦方向のチェック
	int CountH = 0;
	int ColorH = 0;
	save_block();
	combo_check_h(y, x, &CountH, &ColorH);
	if (CountH < 3)
	{
		restore_block();		// ３個未満なら戻す
	}

	// 横方向のチェック
	int CountW = 0;
	int ColorW = 0;
	save_block();
	combo_check_w(y, x, &CountW, &ColorW);
	if (CountW < 3)
	{
		restore_block();
	}
```

```
	// 3つ以上で並んでいるか？
	if ((CountH >= 3 || CountW >= 3))
	{
		if (CountH >= 3)
		{
			Item[ColorH - 1] += CountH;
			Stage_Score += CountH * 10;
		}
		if (CountW >= 3)
		{
			Item[ColorW - 1] += CountW;
			Stage_Score += CountW * 10;
		}
		ret = TRUE;
	}

	return ret;
}

/*******************************
* ステージ制御機能：連鎖チェック処理（縦方向）
* 引　数：なし
* 戻り値：連鎖有無（0:無し　1:有り）
*******************************/
void combo_check_h(int y, int x, int* cnt, int* col)
{
	int Color = 0;
	//対象のブロックが外枠の場合はreturnで処理を抜ける
```

```
	if (Block[y][x].image == 0)
	{
		return;
	}
	*col = Block[y][x].image;
	Color = Block[y][x].image;
	Block[y][x].image = 0;
	(*cnt)++;

	if (Block[y + 1][x].image == Color)
	{
		combo_check_h(y + 1, x, cnt, col);
	}
	if (Block[y - 1][x].image == Color)
	{
		combo_check_h(y - 1, x, cnt, col);
	}
}
/*******************************
* ステージ制御機能：連鎖チェック処理（横方向）
* 引　数：なし
* 戻り値：連鎖有無（0:無し　1:有り）
*******************************/
void combo_check_w(int y, int x, int* cnt, int* col)
{

	int Color = 0;
	//対象ブロックが外枠の場合returnで処理を抜ける
```

```
	if (Block[y][x].image == 0)
	{
		return;
	}

	*col = Block[y][x].image;
	Color = Block[y][x].image;		//色取得
	Block[y][x].image = 0;
	(*cnt)++;

	if (Block[y][x + 1].image == Color)
	{
		combo_check_w(y, x + 1, cnt, col);
	}
	if (Block[y][x - 1].image == Color)
	{
		combo_check_w(y, x - 1, cnt, col);
	}
}

/*******************************
* ステージ制御機能：ブロック情報の保存処理
* 引　数：なし
* 戻り値：なし
*******************************/
void save_block(void)
{
	int i, j;
```

```
	for (i = 0; i < HEIGHT; i++)
	{
		for (j = 0; j < WIDTH; j++)
		{
			Block[i][j].backup = Block[i][j].image;
		}
	}
}

/*******************************
* ステージ制御機能：ブロック情報を戻す処理
* 引　数：なし
* 戻り値：なし
*******************************/
void restore_block(void)
{
	int i, j;

	for (i = 0; i < HEIGHT; i++)
	{
		for (j = 0; j < WIDTH; j++)
		{
			Block[i][j].image = Block[i][j].backup;
		}
	}
}
```