作品(パズルゲーム)について

2019/1/21

大坂泰樹

目次

1.本作品のアウトライン

今回作成したパズルゲームは，山田巧氏が開発した「DX ライブラリ」を用いて，C++言語で制作しました.

「DX ライブラリ」は，DirectX および Windows API を意識せずに使えることをコンセプトとしており，アセットを利用することで，プログラミングの知識が無くても簡単にゲームが開発できる Unity 等のゲームエンジンとは異なり，ゲームの根幹となる部分から細かい動作まで，自力で実装していく必要があります．また，「DX ライブラリ」は，C 言語または C++言語の両方で使用できるように工夫されている一方，C 言語または C++言語の理解が求められます.

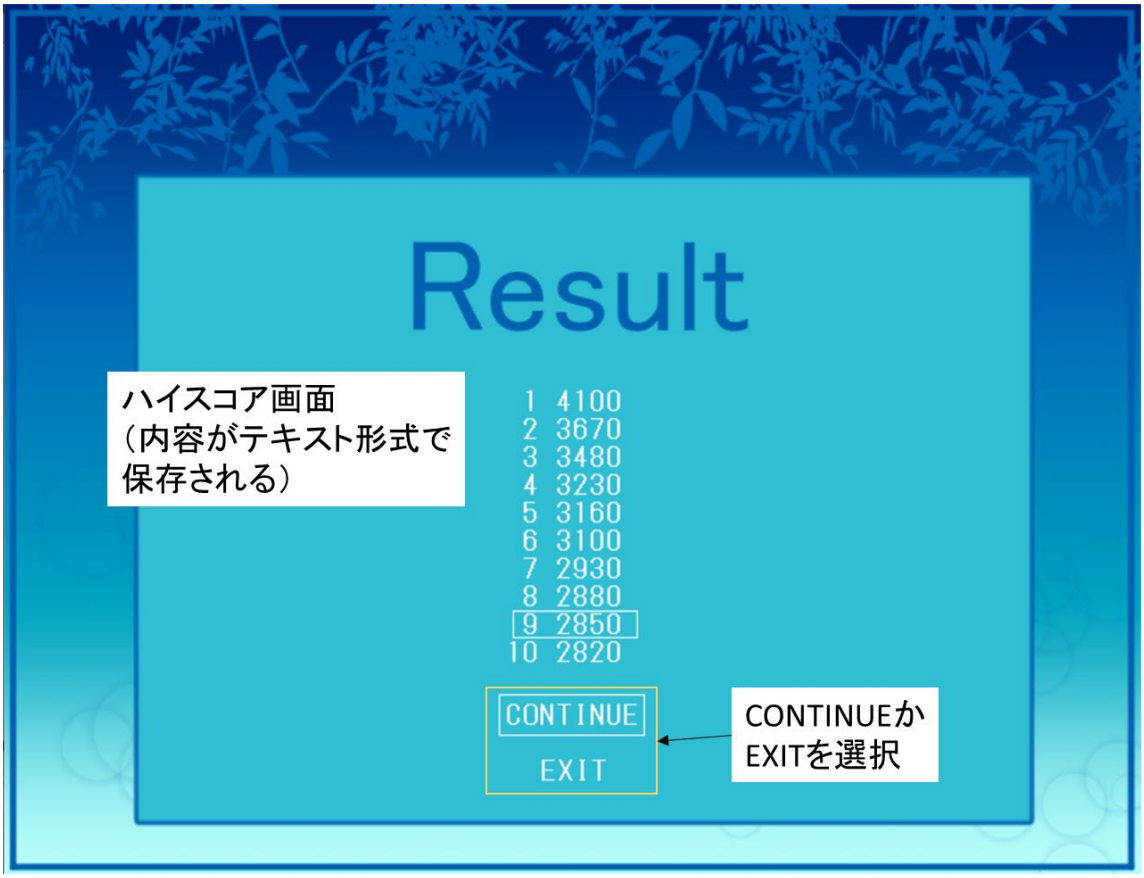
本作品の仕様についてですが，「マッチ 3」という形式のパズルゲームです．「マッチ 3 」とは，盤面上の隣り合うタイルをスワップ（交換）し，同じ種類のタイルが縦または横に 3 つ以上並んだ時にそれらのタイルが消去され，その上にあるタイルが不足分落下し，ポイントを獲得するゲームです．制限時間以内に沢山のタイルを消去するタイプと，決められたスワップ回数内でノルマを達成するタイプの 2 種類の「マッチ 3 」が存在しますが，本作品では前者を採用しました．制限時間は 30 秒とし，気軽に遊べるように設計しました．タイル 1 つにつき 10 ポイントです.

また，「マッチ 3」には，特殊タイルというものが存在します．特殊タイルは，行消しタイプと列消しタイプ，範囲消しタイプの 3 種類存在します．これらのタイルは，通常のタイルと同様に消去でき，そのタイルを消去した時に効果を発揮します．今回は行消しタイプと列消しタイプを実装しました．行消しタイプは，そのタイルを含む 1 行全てを消去し，列消しタイプは，そのタイルを含む 1 列全てを消去します．

例えば，下図の黄色い矢印のタイルをスワップした時，緑のタイルが 3 つ揃い，消去されます．この時，緑のタイルの中に特殊タイルが含まれる為，特殊タイルの効果により，そのタイルを含む 1 行全て（図の赤枠部分）が消去されます．列消しタイル（図中とは異なる）の場合も同様に，1 列を消去します．

特殊タイルの出現確率は，行消しタイプ，列消しタイプ，それぞれ 1/50 となっており，程よく出現する確率となっております（何度も検証を重ねた結果，盤面の大きさ(10×10)に対し，1/50 の確率とすると，程よい出現確率となることを確認済み）．

制限時間の 30 秒が経過すると，ハイスコアが更新され，自動で記録が保存されます．（取った点数は白枠で囲まれている．ランク外の時，白枠は表示されず，1 位のとき，右側に”New Record”と表示される）ハイスコアを確認後，”CONTINUE”及び”EXIT”が選択でき，”CONTINUE”を選択時，盤面がリセットされた上でゲームが再開します．”EXIT”を選択時，最初のメニュー画面に戻ります．メニュー画面に戻った際も，ゲームの盤面はリセットされるようにすることで，スムーズにかつ何度もプレイできるように設計しました．



また，ハイスコアを実装することで，今回は思ったより低い点数を取ったとしても，明確な目標ができ，次はさらに高い点数を狙うことができます．ハイスコアは，高い点数は記録されますが，低い点数は下から順に削除されていきます．ですから，過去の自分の結果と勝負することができ，プレイヤーに合わせた難易度（目標点数）でプレイできます．

本作品の大まかな仕様は以上となります．

２.工夫した点について

1.ルールデータクラスの実装

ルールデータクラスを実装することにより，タイルの消去判定範囲（縦に３つまたは横に３つ）に幅を持たせ，汎用的に表現することにより，変更・追加が効くように設計しました．例えば，マッチ３を桂馬飛びのような配置にすることもでき，盤面全てをマッチングパターンとすることもできます（現実的ではないですが）．マッチ４にすることもできます．ただし，色の指定ができていないので，例えば（赤赤青）の場合は消去対象となる，のような指定はできません．全て同じ色であることが前提です．

また，盤面を初期化時，ルールデータのタイル消去パターンを解りやすくなるように工夫しました．

```
//ボードクラス
Board::Board() {

    //ボードの初期化
    init();

    //ルールデータの初期化
    rule.vanish = std::vector<RuleSub>( 2 ); //消去ルールは2つ存在

    //ルールその1：縦３つ揃った時消去する( X, Y )
    rule.vanish[0].point = { point1<int>( 0, 0 ), point1<int>( 0, 1 ), point1<int>( 0, 2 ) };

    //ルールその2：横３つ揃った時消去する( X, Y )
    rule.vanish[1].point = { point1<int>( 0, 0 ), point1<int>( 1, 0 ), point1<int>( 2, 0 ) };

    //クイックマッチ3でタイルを準備完了状態にする
    while (!qend) {

        //クイックマッチ3の実行準備
        qend = true;

        //クイックマッチ3を実行 ( 成功したらqend = true )
        update_board();
    }
    //スコアリセット
    score = 0;

    //クイックマッチ3完了
    quick = false;

    //ハイスコアのロード
    load_hiscore();
}
```

point1<int>型というのは，独自に作成したデータクラスです．ｘとｙの２つのパラメータを持ち，ゲームを制作する際に，画面に対するオブジェクトの座標を表現する，２次元ベクトルを表現し，加算や乗算，内積の計算などを視覚的に解りやすくする，といった目的で作成しました．今回はタイルの消去パターンを表現するために使用しました．

例えば，縦消去の場合，１つ目の定義(0, 0),(0, 1),(0, 2)を１固まりとし１タイルずつずらしながら全領域を探索することで，「縦に３つ以上揃った時に消去する」を実現することができます．

他にも point2<int>型というクラスがあり，こちらは矩形領域または線分を表現することができ，点と線分の距離も容易に求めることができますが，本作品では使用していません．

2.リソースクラスの実装

ゲームを制作する上で必ず必要となる(※1)要素が，リソースです．リソースは，DX ライブラリの関数でロードされ，返却されたハンドルにより管理されています(※2)．しかし，リソースが多くなると，ハンドルの数も多くなり，描画する際に毎回ハンドルが格納された変数を探さなくてはならなくなり，製作時間増大の原因になってしまいます．そこで，リソースクラスを実装することにより，リソースを使用箇所に関係なく一括で管理することに成功しました．

```
//リソース管理
class Resource {

public:
    //ロード
    static void Load(
        int(*LoadDivGraph)(const TCHAR *, int, int, int, int, int, int *, int),
        int(*LoadGraph)(const TCHAR *, int),
        int(*LoadSoundMem)(const TCHAR *, int, int),
        const char *folder_name);

    //タイルのグラフィック
    static int TileGraph[GRAPH_DIV_NUM_Y][GRAPH_DIV_NUM_X];

    //特殊タイルのグラフィック(縦横一括タイプ)
    static int STile[2];

    //ボード面のグラフィック (0:背景 1:前景)
    static int BoardGraph[2];

    //タイル消去時効果音
    static int VanishSound;

    //タイムアップ表示
    static int Timeup;

    //メニュー画面
    static int StartMenu;

    //ルール説明画面
    static int Rule;

    //リザルト画面
    static int Result;
};
```

Load メソッドについてですが，関数ポインタを 3 つ引数として受け取っています．これは，DX ライブラリに依存せずに関数を実行する為に，汎用性を持たせて，このような形で実装しました．

※1 現代のゲームは画像や音楽，アニメーションを使った，グラフィカルなゲームが主流であり，その為にリソースを使う必要があるという意味．

※2 DX ライブラリの関数の仕様は右記参照（ https://dxlib.xsrv.jp/dxfunc.html ）

3.メインループの簡略化

DX ライブラリで制作するアプリケーションは，WinMain がエントリポイントとなる為長い表記になり，さらにメインループでは，例えばデバッグ操作でスクリーンをクリアせずに動作確認したい場合などに，コメントアウトしたい関数を選択的にコメントアウトしなくてはならず，非常に手間がかかります．そこで，WinMain 関数をプリプロセッサで”main”と名前を定義し，メインループを，フラグ処理により管理するよう，工夫しました．そのようにすることで，メイン関数を簡略化し，メインループで関数を実行する/しないを容易に選択できるようにすることに成功しました．

メインループで毎回実行する関数は下記のとおりです．

・ScreenFlip()（バックグラウンドに描画した内容を表に反映する）

・ProcessMessage()（ウィンドウズメッセージ処理を行う）

・ClearDrawScreen()（画面をクリアする）

これらを実行する/しないを選択する為，以下の定数を定義しました．

・ALL（0x08） ⇒ 0b 0000 1000

・FLIP （0x0E）⇒ 0b 0000 1110

・MESSAGE（0x0D）⇒ 0b 0000 1101

・CLEAR（0x0B）⇒ 0b 0000 1011

これらはフラグを有しており，例えば，CLEAR のみ実行しない場合，while 文の条件式に与える定数として，ビット単位の論理積を用いて，

FLIP & MESSAGE （=0x0C)

のように記述することで，これを実現できます．全てを実行したい場合は，

FLIP & MESSAGE & CLEAR (= 0x08)

と記述しても良いですが，通常は

ALL

と記述するだけで実行できるように，ソースの可読性にも配慮しています．

３.努力した点について

1.メニュー選択直後の挙動について（品質）

本作品の仕様では，ドラッグ中のタイルは左クリックを押している間だけマウスポインタの位置に表示されます．ですから，最初のメニュー画面で”START”をクリックした瞬間に左クリックが押ささっている為タイルを既に持っている状態となり，プレイヤーの意図しない挙動になってしまいます．

そこで，いくつか解決策が挙がりました．メインループの内部の関数呼び出し順を変えようか悩みましたが，リザルト画面からメニュー画面に戻ることがあることと，ソースの見やすさと挙動の両立を考えた結果，デタッチフラグ（マウスの左クリックを離した瞬間のみ ON になるフラグ）を検出し，そのフラグが ON になったときに盤面を動かせるように設計しました．そのようなコーディングにより，見やすいコードが書けただけでなく，プレイヤーの意図しない挙動を防ぐことに成功しました．

2.クイックマッチ３（ゲーム性）

パズルゲームということもあり，タイルはランダムに配置されます．しかし，そのランダム性故，盤面の初期化時に，マッチング条件が揃ってしまうことがあります．つまり，例えば青タイルが３つ横に並んだ状態でゲームが開始することがあります．そのような状態でゲームを開始すると，プレイヤーが操作していないにも拘らずマッチングし，最悪の場合，そのまま連鎖して高得点を手にすることになってしまいます．

それを防ぎ，公平にプレイしていただく為に，「クイックマッチ３」を実装しました．通常，「マッチ３」は人間の目で追えるように，アニメーションを動かしながらタイルを消去するために消去次元の時間を動かし，一定時間経過後，消去するようになっています．消去後，落下タイルの予想経過時間も計算して実際に落下次元の時間も動かします．消去次元と落下次元の２つの時間の進行を，アニメーション処理として組み込んでいるので，一連の動作に時間がかかるのですが，「クイックマッチ３」は，その時間がかかるアニメーション処理を一切省き，必要な動作のみを高速で安定するまで繰り返しています．つまり人間の目では追うことができません．この技術を利用し，初期化時に「クイックマッチ３」を行うことで，ゲーム開始時に突然マッチングが始まるという現象を防いでいます．（具体的には qend が true の時安定状態）

```
//クイックマッチ3でタイルを準備完了状態にする
while (!qend) {

    //クイックマッチ3の実行準備
    qend = true;

    //クイックマッチ3を実行（ 成功したらqend = true ）
    update_board();
}
//スコアリセット
score = 0;

//クイックマッチ3完了
quick = false;
```

3.時間経過後の挙動（情報）

本パズルゲームは，30 秒という制限時間が設けられており，（時間内に消去したタイルの数）×10 ポイントが加算されます． 30 秒経過後はリザルト画面に映るのですが，所謂「落ちもの系」のパズルは，少ない操作で多くの連鎖をすることが醍醐味であり，時間が経過したからといって，連鎖の途中でバッサリと終了してしまうのは，ゲームとしての面白さを半減してしまいかねません．

そこで，連鎖をしている途中かどうかを判断するために，消去次元と落下次元の時間がそれぞれ非 0 である内は連鎖中であるという判断基準で，リザルト画面に移るかどうかを判断するようにしました（30 秒経過後の操作はできません）．さらに，連鎖をしている最中は時間切れであることに気づかない可能性がある為，ゲーム開始から 30 秒経過後には連鎖しているか，していないかに関わらず，”TIME UP”の表示をするようにしました．

連鎖中であることの判断基準と，時間切れであることの情報の表現方法に悩みましたが，編み出すことに成功しました．

４.まとめ

今回作成したゲーム作品についてですが，ゲームをより楽しんで戴けるように，以下の点に気を付けて製作致しました.

① シンプルな操作性
② 制限時間 30 秒という，繰り返し遊ぶために程よい時間
③ ハイスコアの実装による，目標要素の設定

上記要素により，操作は単調ですが直観的で，長時間遊んでも苦痛にならずに，さらに個人のレベルに合わせた目標で再挑戦できるため，”CONTINUE”をつい押してしまう．結果として長時間楽しんで戴ける．そんなゲームができたと思います.

感想ですが，今回はパズルゲームを制作してみて，ゲームを面白いと思わせるような工夫（設計）が思った以上に大変で苦労しました．普段から趣味で行っているゲームプログラミングでは，面白いゲームを作りたいという意思があり技術にも自信がありましたが，自分以外の目線でゲームを見たときにそのような工夫が少し足りない部分があり，製品としては少し物足りない印象がありました．プロとしてのゲーム制作を意識して制作に取り掛かることで，製品としてのゲーム制作が何たるかを理解しました．実際にプロの現場で携わらせていただく中で，担当部品が割り当てられていくと思いますが，今回の経験を活かし，常に全体像を捉えながら，自分が今どの部品を作成しているのか，どうしたら面白い製品ができるのかをいつも以上に意識して取り組んでいこうと思います．ゲームとしての面白さだけでなく，どうすれば売り上げに繋がるのか，という考え方もプロとして必要な概念であるため，これから経験を積んでいく上で身に付けていこうと思います.

最後に余談ですが，私は将来 VR による社会的インフラを構築していきたいという目標があります．学校や職場等の環境に VR 技術を取り入れることで物理的移動時間の短縮や交通事故の回避が可能になり，さらに現実世界では実現不可能な体験を，VR 空間内でなら実現可能です（逆も然りですが）．その為にも，ゲーム業界で技術を磨き，目標達成に向けて努力をしていきたいと考えております.

以上，お読み戴きありがとうございます.