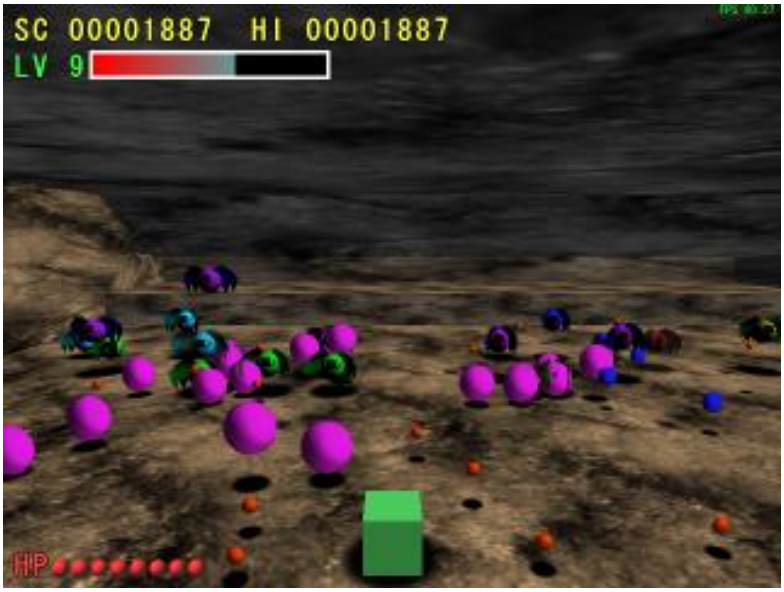
後期課題2

3DPG 「課題制作その1（STG）」



イメージ図

■ゲーム仕様

1)インベーダー形式のシューティングゲームでプレイヤーを動かして弾を発射して敵を倒します。

2)プレイヤー

- 3D モデルかプリミティブ形状を使用する。
- 「←」「→」キーで左右移動。 ※「↑」「↓」キーでの前後移動は任意。
- 「SPACE」or「左 Ctrl」などで前方に弾を発射する。
- プレイヤーは体力があり、体力がなくなるとゲームオーバーとなる。

3)プレイヤー弾

- 3D モデルかプリミティブ形状を使用する。
- 基本はプレイヤーの正面方向に飛んでいく。
- 奥行の限界点に来ると弾が消滅する。
- 敵に当たると敵にダメージを与え弾が消滅する。
- 適度に連射ができる様にする。

4)敵

- 3D モデルを使用する。
- ランダムにフィールド上に出現してフラフラと浮遊する。
- 適当なタイミングで弾を発射する。（正面方向、3WAY、プレイヤの方向など）
- 色を変えて複数の種類を作成する。
- 体力を持たせる。

5)フィールド&カメラ

・3D モデルを使用する。

・カメラは基本的に固定で良いが、プレイヤに追従して左右移動しても良い。

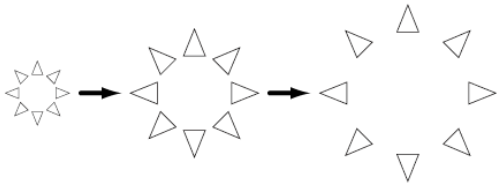
6)その他

・画面サイズは（横 1024、縦 600）とする。

・スコア表示を行う（2D で良い）。

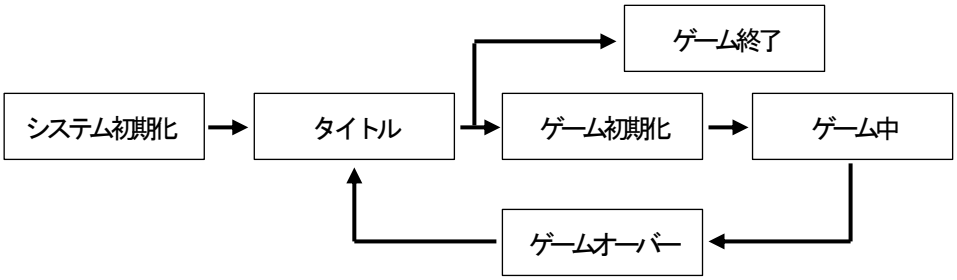
・プレイヤーの体力を表示する。

・弾が敵に当たる時などにエフェクト効果を付ける。サウンドも鳴らす。



エフェクトの例）
発生源からポリゴンが放射状に広がって次第に消える

7)画面遷移



・システム初期化・・・・・DxLib の初期化などのゲーム本体に関わる初期化を行う。

・タイトル画面・・・・・・・・真っ暗画面は NG。ゲームのデモなど流すと良い。
※何のキーで始まるかを表記する。

・ゲーム初期化・・・・・・・インスタンスの管理をしっかり行う。
※コンストラクタでインスタンス数を加算する。

・ゲーム中・・・・・・・・・・・ゲームとして成立する様に、敵の動きや攻撃などに気を配る事。
※インスタンス数を常に画面上部に表示する。

・ゲーム終了・・・・・・・・・インスタンスの解放処理をしっかり行う。
※デストラクタでインスタンス数を減算する。

・ゲームオーバー・・・・・・終了した状態を保持していつやられたかが分かる様にしておく。
※何のキーでタイトルに戻るかの表記をする。

■クラス構成

1)main.cpp ・・・・・・・・・・・ WinMain()のエントリーポイント。GameTask を呼び出す。
2)GameTask クラス ・・・・ 画面遷移を管理するクラス ※シングルトンクラス
3)KeyMng クラス ・・・・・・ キー入力用の管理クラス ※シングルトンクラス
4)ImageMng クラス ・・・・ 2D 画像用の管理クラス ※シングルトンクラス
5)Camera クラス ・・・・・・ カメラ管理用。
6)Player クラス ・・・・・・・ プレイヤー管理用。
7)Enemy クラス ・・・・・・ 敵キャラ管理用。
8)Shot クラス ・・・・・・・・ プレイヤー弾管理用。
9)Bullet クラス ・・・・・・・ 敵弾管理用。
10)Effect クラス ・・・・・・ エフェクト管理用。
11)Field クラス ・・・・・・・ 背景管理用。
12)HitCheck クラス ・・・ 当たり判定管理用。
13)その他、必要に合わせて追加を行う。

■評価ポイント

□クラス構成が指示通りにできている。(10.点)
□画面遷移が指示通りにできている。(10.点)
□プレイヤーが指示通りにできている。(10.点)
□敵キャラクターが指示通りにきている。(10.点)
□プレイヤー弾が指示通りにできている。(10.点)
□敵弾が指示通りにできている。(10.点)
□エフェクトが指示通りにできている。(10.点)
□キー入力・2D 画像処理はマネージャークラスでの実装ができている。(10.点)
□全てのインスタンスについて加算。減算処理がされ画面表示ができている。(10.点)
□完成されたゲームとして遊べるようになっている。(10.点)

■提出期限・場所

期限:平成 30 年 11 月 2 日(金) 19:00
場所:¥¥stfs¥APC_ABCC クリエイティブ¥gakuseigame¥2 年生 classC¥課題 2_11 月提出
提出内容:それぞれ名前のフォルダを作成し、その中にプロジェクトフォルダを保存する。
※「debug」「.vs」フォルダは削除しておく事。

□資料

■メモリーリークチェックの仕込み

メモリーリークチェックとは、確保したメモリを確実にクリアしているかどうかを確認する事です。
オブジェクトを new してそのまま終了すると、そのオブジェクト分のメモリがどこかに確保されたままになって使用できるメモリの領域を圧迫していきます。
確保したメモリは責任を持ってクリアしなければなりません。この課題では、手動でメモリの解放を行う事でどのようにプログラムを動かせばいいかを学んでいきます。

1)どのように確認するのか？
実際には「new したものは必ず delete する」を徹底する事で実現できますので、new した数と delete した数をカウントして 0 になっていればできている事が確認できます。

カウント確認の仕込み

```
class GameTask
{
public:
    int playerObjectCnt;   // プレイヤーカウント
    int shotObjectCnt;     // ショットカウント
    int enemyObjectCnt;    // 敵カウント

    void addPlayerObjectCnt();    // プレイヤーカウント追加
    void removePlayerObjectCnt(); // プレイヤーカウント削除
    void addEnemyObjectCnt();     // 敵カウント追加
    void removeEnemyObjectCnt();  // 敵カウント削除
    void addShotObjectCnt();      // ショットカウント追加
    void removeShotObjectCnt();   // ショットカウント削除
};
```

```
void GameTask::addPlayerObjectCnt()
{
    playerObjectCnt++;
}
void GameTask::removePlayerObjectCnt()
{
    playerObjectCnt--;
}
```

※敵、ショットも同様

状態の表示(どの画面遷移でも描画を行う)

```
int GameTask::Update()
{
   // 省略

    DrawFormatString(0, 32, 0xffffff, "playerObjectCount = %d", playerObjectCnt);
    DrawFormatString(0, 48, 0xffffff, "enemyObjectCount  = %d", enemyObjectCnt);
    DrawFormatString(0, 64, 0xffffff, "shotObjectCount   = %d", shotObjectCnt);

    return 0;
}
```

2)実際の作業　例)プレイヤー

コンストラクタ　→　カウントを追加

```
Player::Player()
{
    Init();
    GameTask::GetInstance().addPlayerObjectCnt();  // ﾌﾟﾚｲﾔｰｶｳﾝﾄ+1
}
```

デストラクタ　→　カウントを削除

```
Player::~Player()
{
    GameTask::GetInstance().removePlayerObjectCnt();  // ﾌﾟﾚｲﾔｰｶｳﾝﾄ-1
}
```