

ポートフォリオ

学校法人情報文化学園アーツカレッジヨコハマ

ゲームクリエイター学科 プログラマーコース

K225003 伊藤 大翔

目次

- 自己紹介
- LOST RECOLLECTION・配置ツール
 - 概要
 - あたり判定
 - ツール制作理由
 - ツール機能
 - 設計
 - 追加した点
 - 苦労した点
 - 今後について
- ユカオトセ！
 - 概要
 - コンテストについて
 - こだわり

自己紹介

趣味

- ・ 鉄道に乗る
- ・ 写真を撮る
- ・ 散歩

受賞歴

LABコン2023 商品スポンサー賞
LABコン2023 来場者投票数最多賞

(LABコン詳細→

<https://techplay.jp/event/887201>)

-----学内-----

1年生

2学期企業連携講座 優秀作品
3学期企業連携講座 優秀作品
GS AWARD 受賞

2年生

前期企業連携講座 優秀作品
後期企業連携講座 優秀作品
GS AWARD 受賞

好きなゲーム作品



BRAVELY DEFAULT



マインクラフト



原神



モンスターハンターワールド
(アイスボーン)



ポケットモンスター
ブラック2・ホワイト2



とびだせ
どうぶつの森



スマブラSP



カービィのエアライド

など

自分のアピールポイント・楽しむ力

ゲーム制作で大事なことは、ゲームが面白い、楽しいことだと考えています。自分は作品制作、アルバイトなども含めた日常生活を全力で楽しみ、常に面白いことがないかを探しています。また、授業で使用したソースコードを改造し、面白いものや、より良いものにしていました。

使える言語・ソフト

C/C++ C#(少し)
Visual Studio Unity CakeWalk
PowerPoint Excel Word
PremierePro
Blender

作品概要・LOST RECOLLECTION



制作環境 : Visual Studio2019/C++
TortoiseSVN, DxLib, ImGui

ゲーム制作人数 : プログラマー3人・デザイナー1人
担当箇所 : オブジェクト全般・あたり判定

ゲームタイトル : LOST RECOLLECTION

ゲーム概要 : ホラー調のステルスゲームです。
廃墟となった遊園地で、突如動き出した着ぐるみマスコットたちに捕まらない様に脱出を目指します。
ゴミ箱を被ることで見つかりにくくなります。

担当詳細 : オブジェクトの配置ツールと、オブジェクトとキャラクターのあたり判定、プレイヤーの一部機能の追加を行いました。

あたり判定

```
struct UniformityCollider {  
    /// <summary>  
    /// キューブのデータをセットする  
    /// </summary>  
    /// <param name="cube"></param>  
    void SetData(const CubeCollider& cube);  
  
    /// <summary>  
    /// カプセルのデータをセットする  
    /// </summary>  
    /// <param name="capsule"></param>  
    void SetData(const CapsuleCollider& capsule);  
  
    /// <summary>  
    /// 線分のデータをセットする  
    /// </summary>  
    /// <param name="start"></param>  
    /// <param name="end"></param>  
    void SetData(const VECTOR& start, const VECTOR& end);  
  
    VECTOR pos = VZERO;  
    VECTOR axis[AXIS_NUM] = { VZERO, VZERO, VZERO };  
    VECTOR normAxis[AXIS_NUM] = { VGet(1,0,0), VGet(0,1,0), VGet(0,0,1) };  
    float radius = 0; //半径  
    float farthestLength = 0; //中心点から最も遠い座標  
};
```

共通コライダーの構造体

```
bool CheckHit(const CubeCollider& cube1, const CubeCollider& cube2, VECTOR* sub = nullptr);  
bool CheckHit(const CubeCollider& cube, const CapsuleCollider& capsule, VECTOR* sub = nullptr);  
bool CheckHit(const CapsuleCollider& capsule, const CubeCollider& cube, VECTOR* sub = nullptr);  
bool CheckHit(const CubeCollider& cube, const SphereCollider& sphere, VECTOR* sub = nullptr);  
bool CheckHit(const SphereCollider& sphere, const CubeCollider& cube, VECTOR* sub = nullptr);  
bool CheckHit(const CapsuleCollider& capsule1, const CapsuleCollider& capsule2, VECTOR* sub = nullptr);  
bool CheckHit(const CapsuleCollider& capsule, const SphereCollider& sphere, VECTOR* sub = nullptr);  
bool CheckHit(const SphereCollider& sphere, const CapsuleCollider& capsule, VECTOR* sub = nullptr);  
bool CheckHit(const SphereCollider& sphere1, const SphereCollider& sphere2, VECTOR* sub = nullptr);
```

コライダーを変換せずに呼び出せるように、オーバーロード

```
bool Collision::CheckHit(const CapsuleCo  
{  
    //統一コライダーに変更  
    UniformityCollider c1, c2;  
    c1.SetData(capsule);  
    c2.SetData(cube);  
  
    return CheckHit(c1, c2, sub);  
}
```

最初は個人で制作していましたが、うまくいきませんでした。

クラスメイトで、あたり判定を制作している人に相談したところ、汎用的なものを作ったが、試してくれる人がいないとのことだったので、自分の勉強もかねて、そのプログラムを読み、実際のゲームに反映しました。

共通のコライダーを用意し、0BB、カプセル、球体をそれぞれコライダー内で変換して、判定を行います。

あたり判定

```
//col1の軸で判定を行う
for (int i = 0; i < AXIS_NUM; i++)
{
    VECTOR nAxis = col1.normAxis[i];
    float distance = abs(VDot(posSub, nAxis));

    float len1 = VSize(col1.axis[i]) + col1.radius;

    float len2 = col2.radius;
    for (VECTOR axis : col2.axis)
    {
        len2 += abs(VDot(axis, nAxis));
    }

    if (distance > len1 + len2)
    {
        return false;
    }

    if (VSize(colSub) > (len1 + len2) - distance)
    {
        colSub = nAxis * ((len1 + len2) - distance);
    }
}
```

分離軸を用いた判定の一部

```
//互いの線分の最近点間のベクトルと長さで判定
Line line1, line2;
line1.SetData(capsule1);
line2.SetData(capsule2);

VECTOR shortest = ShortestVector(line2, line1);
float lenge = VSize(shortest);

if (leng > radius1 + radius2)
{
    return false;
}
```

```
struct Line {
    void SetData(const CapsuleCollider& capsule);
    void SetData(const VECTOR& start, const VECTOR& end);

    VECTOR start = VZERO;
    VECTOR end = VZERO;
    VECTOR direction = VGet(0, 0, 1);
};

/// <summary>
/// 点から線分の最近点へのベクトル
/// </summary>
/// <param name="point">点</param>
/// <param name="line">線分</param>
/// <returns>点から線分の最近点へのベクトル</returns>
VECTOR ShortestVector(const VECTOR& point, Line line);
```

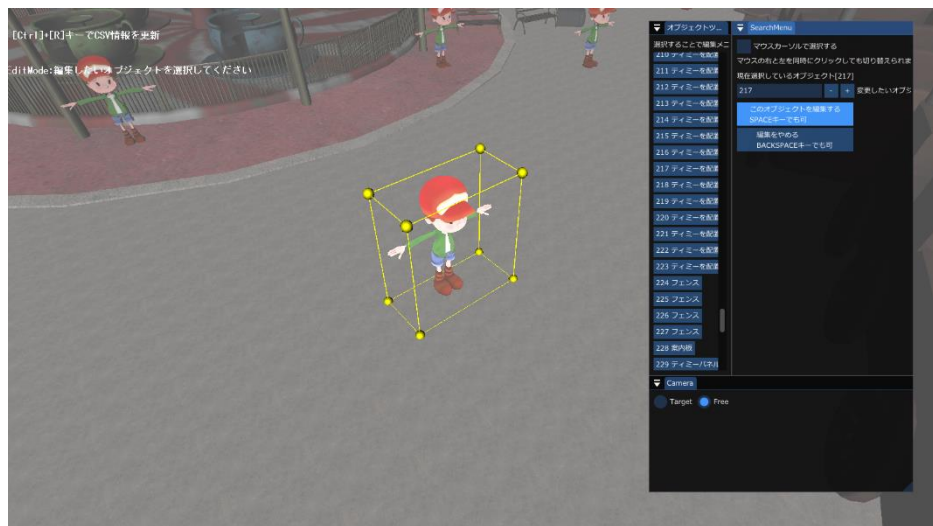
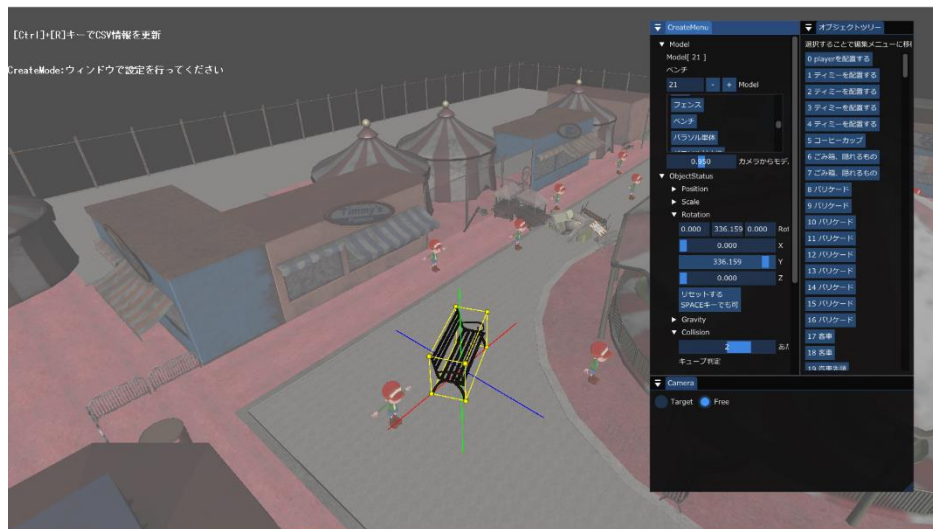
点と点の距離判定の一部と線判定の構造体

共通コライダーは分離軸を用いて判定を行います。

ただし、すべての判定に共通コライダーを使用するわけではなく、球体同士など一部処理では変換を行わずにその場で判定を行います。

また、レイ判定を行うときやカプセル同士の判定を行うときには、線判定の構造体を使用し、処理を簡略化しています。

作品概要・オブジェクト配置ツール



制作環境：Visual Studio2019/C++
TortoiseSVN, DxLib, ImGui

ゲーム制作人数：プログラマー3人・デザイナー1人
担当箇所：配置ツールプログラム全般・あたり判定

ゲームタイトル：LOST RECOLLECTION

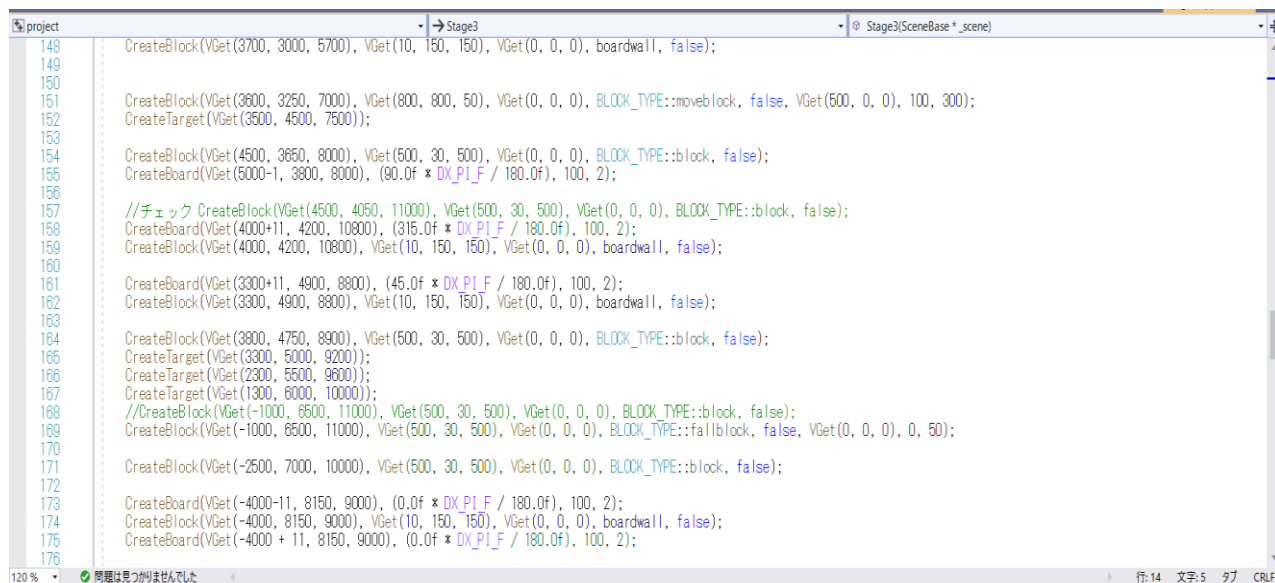
ゲーム概要：ホラー調のステルスゲームです。
廃墟となった遊園地で、突如動き出した着ぐるみマスコットたちに捕まらない様に脱出を目指します。
ゴミ箱を被ることで見つかりにくくなります。

配置ツール概要：

3D空間上でオブジェクトや敵の配置を行うツールです。
配置されたデータはCSVファイルに保存されます。

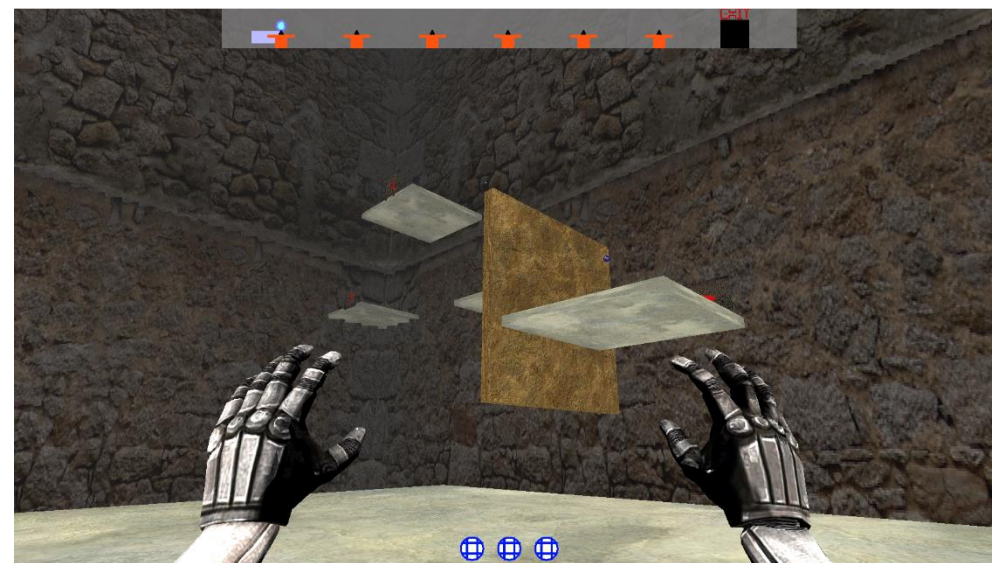
モデルの追加や変更もCSV上でできるようにしました。

制作理由



```
project Stage3
Stage3(SceneBase *_scene)

148 CreateBlock(VGet(3700, 3000, 5700), VGet(10, 150, 150), VGet(0, 0, 0), boardwall, false);
149
150
151 CreateBlock(VGet(3600, 3250, 7000), VGet(800, 800, 50), VGet(0, 0, 0), BLOCK_TYPE::moveblock, false, VGet(500, 0, 0), 100, 300);
152 CreateTarget(VGet(3600, 4500, 7500));
153
154 CreateBlock(VGet(4500, 3650, 8000), VGet(500, 30, 500), VGet(0, 0, 0), BLOCK_TYPE::block, false);
155 CreateBoard(VGet(5000-1, 3800, 8000), (90.0f * DX_PI_F / 180.0f), 100, 2);
156
157 //チェック CreateBlock(VGet(4500, 4050, 11000), VGet(500, 30, 500), VGet(0, 0, 0), BLOCK_TYPE::block, false);
158 CreateBoard(VGet(4000+11, 4200, 10800), (315.0f * DX_PI_F / 180.0f), 100, 2);
159 CreateBlock(VGet(4000, 4200, 10800), VGet(10, 150, 150), VGet(0, 0, 0), boardwall, false);
160
161 CreateBoard(VGet(3300+11, 4900, 8800), (45.0f * DX_PI_F / 180.0f), 100, 2);
162 CreateBlock(VGet(3300, 4900, 8800), VGet(10, 150, 150), VGet(0, 0, 0), boardwall, false);
163
164 CreateBlock(VGet(3800, 4750, 8900), VGet(500, 30, 500), VGet(0, 0, 0), BLOCK_TYPE::block, false);
165 CreateTarget(VGet(3300, 5000, 9200));
166 CreateTarget(VGet(2300, 5500, 9600));
167 CreateTarget(VGet(1300, 6000, 10000));
168 //CreateBlock(VGet(-1000, 6500, 11000), VGet(500, 30, 500), VGet(0, 0, 0), BLOCK_TYPE::block, false);
169 CreateBlock(VGet(-1000, 6500, 11000), VGet(500, 30, 500), VGet(0, 0, 0), BLOCK_TYPE::fallblock, false, VGet(0, 0, 0), 0, 50);
170
171 CreateBlock(VGet(-2500, 7000, 10000), VGet(500, 30, 500), VGet(0, 0, 0), BLOCK_TYPE::block, false);
172
173 CreateBoard(VGet(-4000-11, 8150, 9000), (0.0f * DX_PI_F / 180.0f), 100, 2);
174 CreateBlock(VGet(-4000, 8150, 9000), VGet(10, 150, 150), VGet(0, 0, 0), boardwall, false);
175 CreateBoard(VGet(-4000 + 11, 8150, 9000), (0.0f * DX_PI_F / 180.0f), 100, 2);
176
```



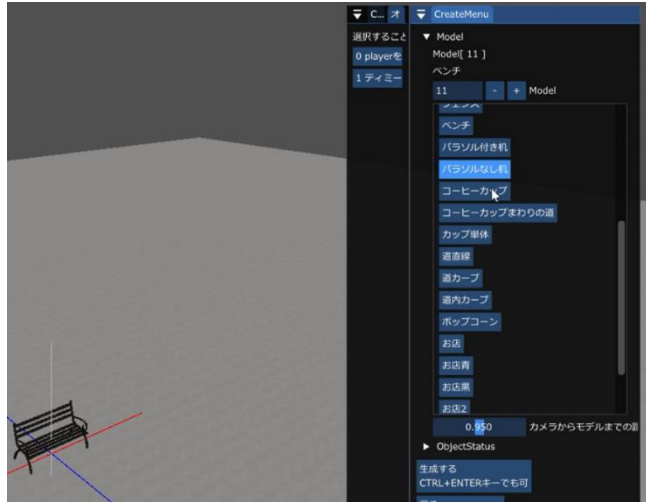
↑ DungeonsWireのステージのソースコード（左）と、そのコードから生成されたブロックたち（右） ↑

2023年のゲーム大賞に作品を提出する際に作ったチーム作品『DungeonsWire』で、自分はステージ制作を担当していました。しかし、ステージ内容をすべてコード内で入力していたため、とても分かりづらく制作中は、ブロックを置いてデバッグビルド、うまくいかなければコード上で修正しもう一度…。という作業を繰り返す手間が発生していました。

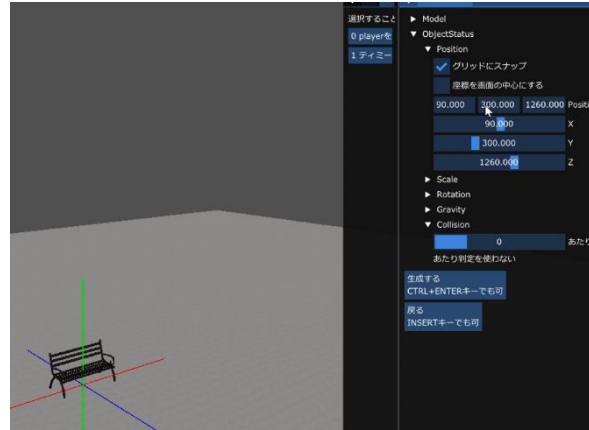
就職活動作品を制作するにあたって、組んだチームに背景志望のCGデザイナーがいたため、そして自分がこの作業をもう行いたくないと思ったため、今回のツールを制作しました。

目標として、デザイナーに使ってもらい、ステージを設計してもらおうという目標を立てました。

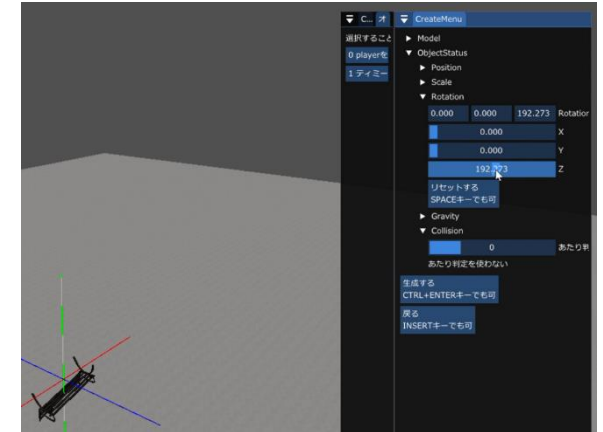
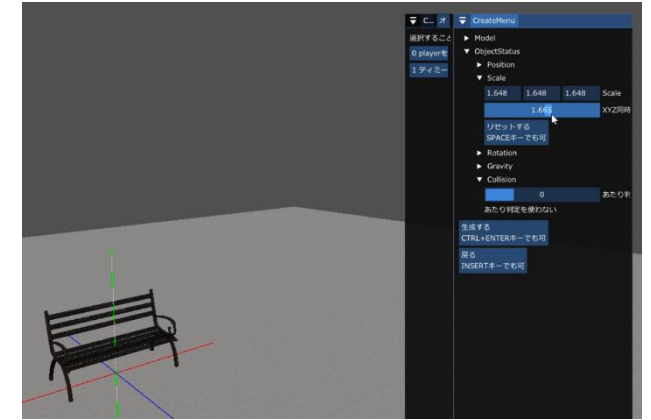
配置の流れ・1



1. モデルを選択

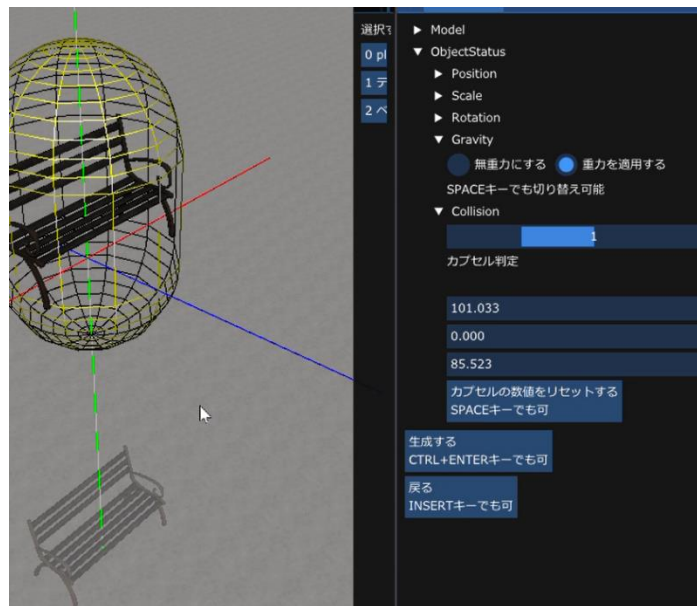


2. 位置(左上)・拡張(右上)・回転(右下)を決定

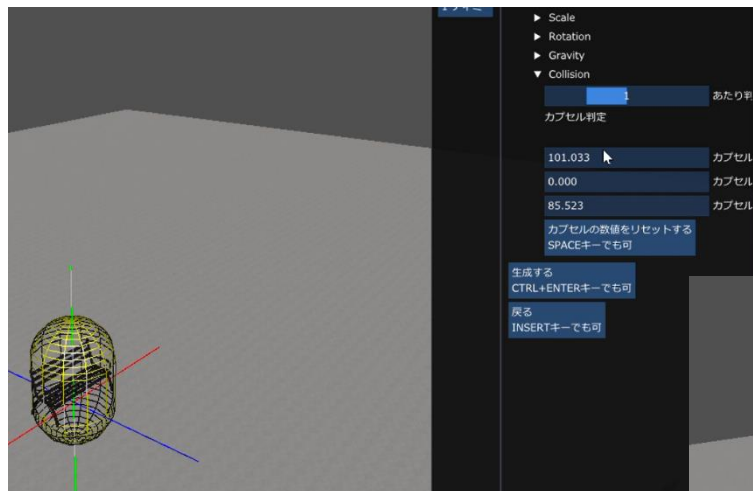


1. モデルを選択します。ImGuiの機能を使い、ツリー状で一覧にしているのでクリックすれば変更されます。
2. 位置を決めます。グリッドスナップ機能を使うと各数値が1の位で四捨五入されるようになります。画面中心機能がついていると、カメラの向いている方向にモデルがついてきます。オフにしている場合、スライダーが表示されます。スライダーの上限下限はステージモデルから自動で取得するようにしています。拡張・回転は、スライダーを用いて設定を行えるようにしたほか、リセットボタンもつけました。

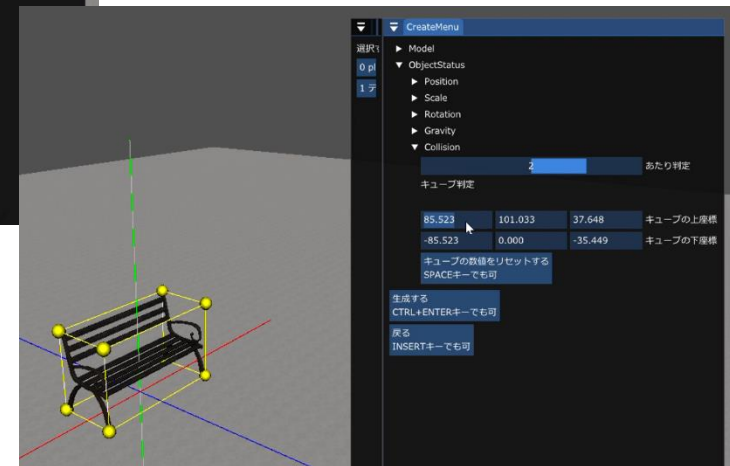
配置の流れ・2



3. 重力の有無



4. あたり判定の有無

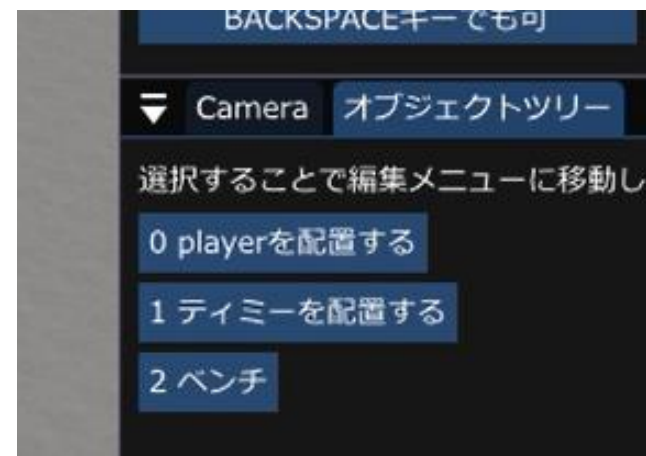


3. 重力を設定します。重力があると生成後、自動で地面に設置されます。
4. あたり判定を設定します。判定なし・カプセル・キューブの3種類に加え、今回のゲームで使用するために、被り物の設定を追加しました。カプセル・キューブは大きさを変更可能にしています。
モデルのメッシュからXYZそれぞれの最大値最小値を取得し、それをもとに判定の初期値を決定しています。
この6項目を設定し、生成するボタンを押すと生成されます。

編集機能



↑ マウスカーソルで選択



↑ ImGui のリスト機能で選択

マウスで選択だけでなく、リストからボタンを押しても編集できるようにしました。
リストから選択した場合、カメラがオブジェクトのもとに移動するようにし、視覚的にわかりやすくしました。
選択後は、生成と同じ項目同じ操作で編集ができます。ただし、生成するボタンは変更確定ボタンに変わり、新しくオブジェクト削除ボタンが増えます。
また、編集メニューを開かずに、選択段階でも削除できるようにしています。

設計・情報保存

```
for (ObjectSet* o : obj) {  
    o->CsvExport();  
}
```

↑ Object1つ1つが、自身の数値をCSVに入力

CsvExport() 関数の処理の一部→

```
std::ofstream writing_file(fileName, std::ios::app);  
std::string str;  
{  
    str = std::to_string(modelNum);  
    str += ",";  
    str += std::to_string(transform.position.x);  
    str += ",";  
    str += std::to_string(transform.position.y);  
    str += ",";  
    str += std::to_string(transform.position.z);  
    str += ",";  
  
    str += std::to_string(transform.scale.x);  
    str += ",";  
    str += std::to_string(transform.scale.y);  
    str += ",";  
    str += std::to_string(transform.scale.z);  
    str += ",";  
  
    str += std::to_string(transform.rotation.x);  
    str += ",";  
    str += std::to_string(transform.rotation.y);  
    str += ",";  
    str += std::to_string(transform.rotation.z);  
    str += ",";  
}
```



CSVが書き換わる→

25	14.99999	0	14.99657	0.52	0.52	0.52	0	0	0	0	1	1000	1650.562	1000	-1000	-72.4755	-1000	907.8053
6	2325.022	0	-2145.02	2.2	1.2	2.2	0	0	0	0	3	42.97654	50.04993	42.97512	-42.9765	-50.0667	-42.9765	64.46481
6	-2069.97	0	1619.978	2.2	1.2	2.2	0	0	0	0	3	42.97654	50.04993	42.97512	-42.9765	-50.0667	-42.9765	64.46481
9	2000	0	-2400	1	1	1	0	0	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	1800	0	-2400	1	1	1	0	0	-0.05236	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	1600	0	-2400	1	1	1	0	-0.31416	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	1400	0	-2400	1	1	1	0	0.122173	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	1200	0	-2400	1	1	1	0	0.261799	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	999.9999	0	-2400	1	1	1	0	0	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	2200	0	-2400	1	1	1	0	-0.29671	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	2399.999	0	-2400	1	1	1	0	0.122173	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	2600	0	-2400	1	1	1	0	0.401426	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
15	239.9997	0	-2400	1	1	1	0	1.204277	0	0	2	221.5502	670.2036	500	-220.327	0	-500	1433.627
15	-709.733	210	-2551.57	1	1	1	0	1.204277	-1.5708	1	2	221.5502	670.2036	500	-220.327	0	-500	1433.627

設計・自動配置

CSV(配置データ) →



25	14.99999	0	14.99657	0.52	0.52	0.52	0	0	0	0	0	1	1000	1650.562	1000	-1000	-72.4755	-1000	907.8053
6	2325.022	0	-2145.02	2.2	1.2	2.2	0	0	0	0	0	3	42.97654	50.04993	42.97512	-42.9765	-50.0667	-42.9765	64.46481
6	-2069.97	0	1619.978	2.2	1.2	2.2	0	0	0	0	0	3	42.97654	50.04993	42.97512	-42.9765	-50.0667	-42.9765	64.46481
9	2000	0	-2400	1	1	1	0	0	0	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	1800	0	-2400	1	1	1	0	0	0	-0.05236	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	1600	0	-2400	1	1	1	0	-0.31416	0	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	1400	0	-2400	1	1	1	0	0.122173	0	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	1200	0	-2400	1	1	1	0	0.261799	0	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	999.9999	0	-2400	1	1	1	0	0	0	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	2200	0	-2400	1	1	1	0	-0.29671	0	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	2399.999	0	-2400	1	1	1	0	0.122173	0	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
9	2600	0	-2400	1	1	1	0	0.401426	0	0	0	1	55	53.99019	30.04246	-55	-25.3068	-24.9575	55
15	239.9997	0	-2400	1	1	1	0	1.204277	0	0	0	2	221.5502	670.2036	500	-220.327	0	-500	1433.627
15	-709.733	210	-2551.57	1	1	1	0	1.204277	-1.5708	1	2	221.5502	670.2036	500	-220.327	0	-500	1433.627	

```
31  /// <summary>
32  /// Object生成時につかうデータ
33  /// </summary>
34  struct ObjectCreateData {
35      int model;           //モデルそのもの
36      int modelNum;      //モデルがCSVの何番目にいるか
37      VECTOR Pos;        //位置座標
38      VECTOR Rot;        //回転
39      VECTOR ScI;        //拡張
40      bool zeroGravity;   //重力を使用するか
41      int hit;           //あたり判定タイプ
42      VECTOR topMesh;    //あたり判定時に使用。XYZ各値で最も数字の大きいもの
43      VECTOR bottomMesh; //あたり判定時に使用。XYZ各値で最も数字の小さいもの
44      float radius;      //カプセルあたり判定の時に使用。円の半径
45  }
```

↑ オブジェクト情報の構造体

```
oed->Pos = VGet(csv->GetFloat(readLine, 1),
               csv->GetFloat(readLine, 2), csv->GetFloat(readLine, 3));

oed->ScI = VGet(csv->GetFloat(readLine, 4),
               csv->GetFloat(readLine, 5), csv->GetFloat(readLine, 6));

oed->Rot = VGet(csv->GetFloat(readLine, 7),
               csv->GetFloat(readLine, 8), csv->GetFloat(readLine, 9));

oed->zeroGravity = csv->IsBool(readLine, 10);

oed->hit = csv->GetInt(readLine, 11);

int n = csv->GetInt(readLine, 0) + 1;
oed->model = ResourceLoader::MVLLoadModel(model->GetString(n, 0));

oed->modelNum = csv->GetInt(readLine, 0);

oed->topMesh = VGet(csv->GetFloat(readLine, 12),
                  csv->GetFloat(readLine, 13), csv->GetFloat(readLine, 14));

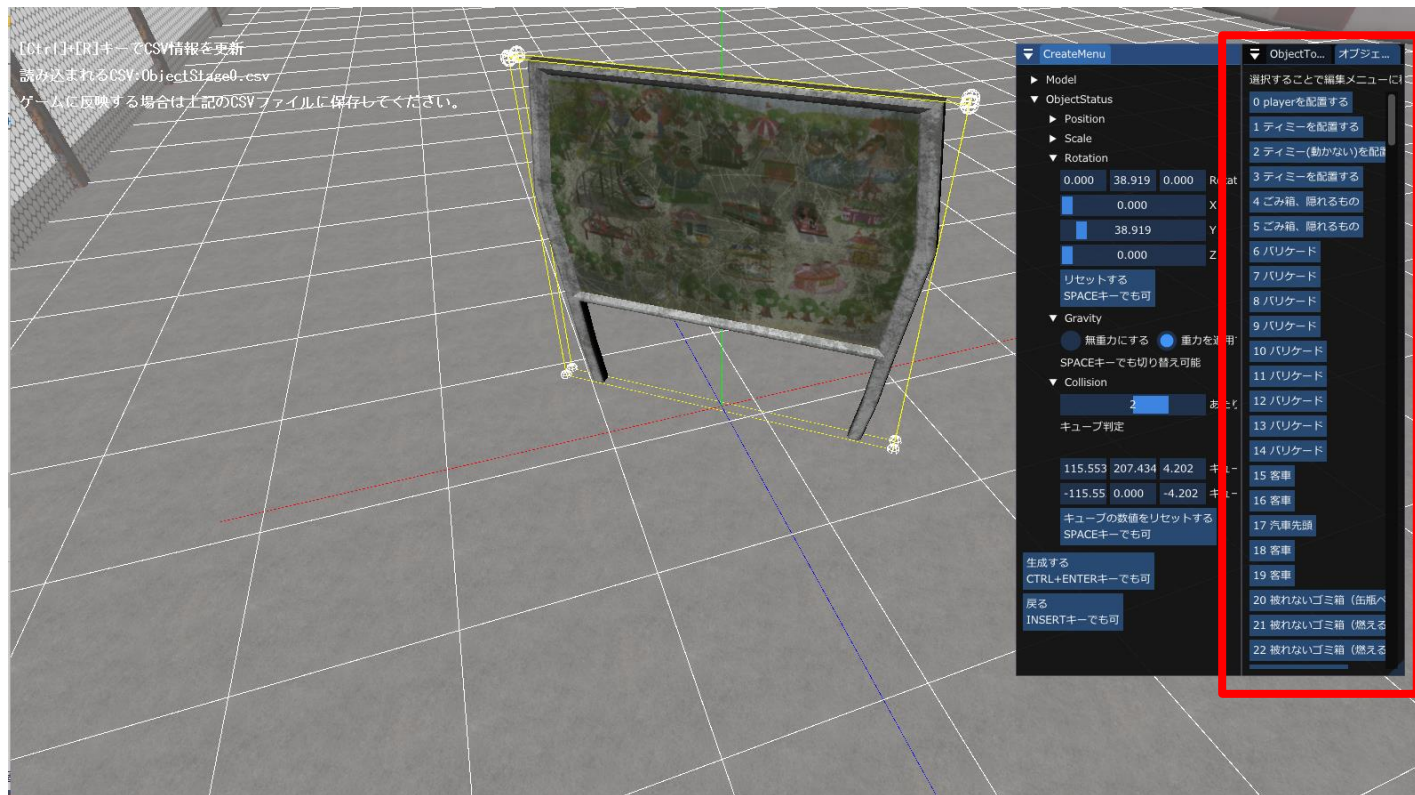
oed->bottomMesh = VGet(csv->GetFloat(readLine, 15),
                      csv->GetFloat(readLine, 16), csv->GetFloat(readLine, 17));
```

↑ オブジェクト情報を格納する処理

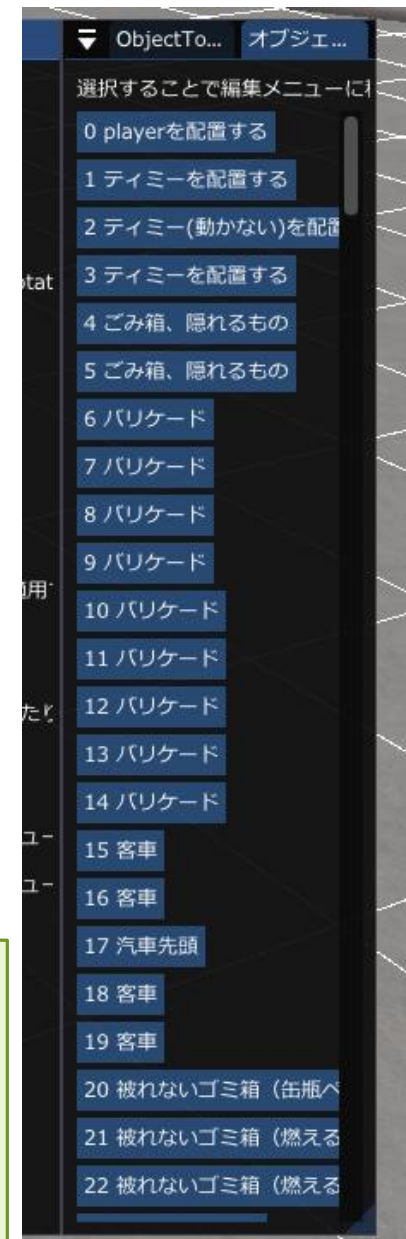
vector配列で生成→

```
void ObjectSetManager::Create(const ObjectCreateData& data)
{
    ObjectSet* newObject = new ObjectSet(data);
    obj.emplace_back(newObject);
}
```


追加した機能 リスト



拡大



実際にチームの人に使用してもらい、フィードバックをもらいました。
その時の要望の一つが、オブジェクトリスト機能です。
オブジェクトの配列を順番に表示して、クリックすることで該当オブジェクトを
編集する画面に直接変更するようになっています。

追加した機能 一つ戻す・やり直す機能

```
std::vector<std::vector<ObjectSet*>>undoArray;//これまでのオブジェクト配置情報  
std::vector<std::vector<ObjectSet*>>redoArray;//undoを使用したときに情報を保持する配列
```

↑ オブジェクトの配置情報を可変長の二次元配列で保存

Ctrl+Z→
Shift+Z→

配置状態を1つ前に戻す
やり直し

←キーボードだけでなく、
ImGuiのボタンでも表示

Ctrl+Z/Shift+ZかImGuiのボタンで配置情報を一つ戻す機能を実装しました。
配置データを二次元配列で保持し、全体の数が増えたり、座標などのデータが変更されたときに
データが追加されるようにしました。
シーンが変更されたときに、保持している情報をリリースするようにしています。

苦労した点・UI

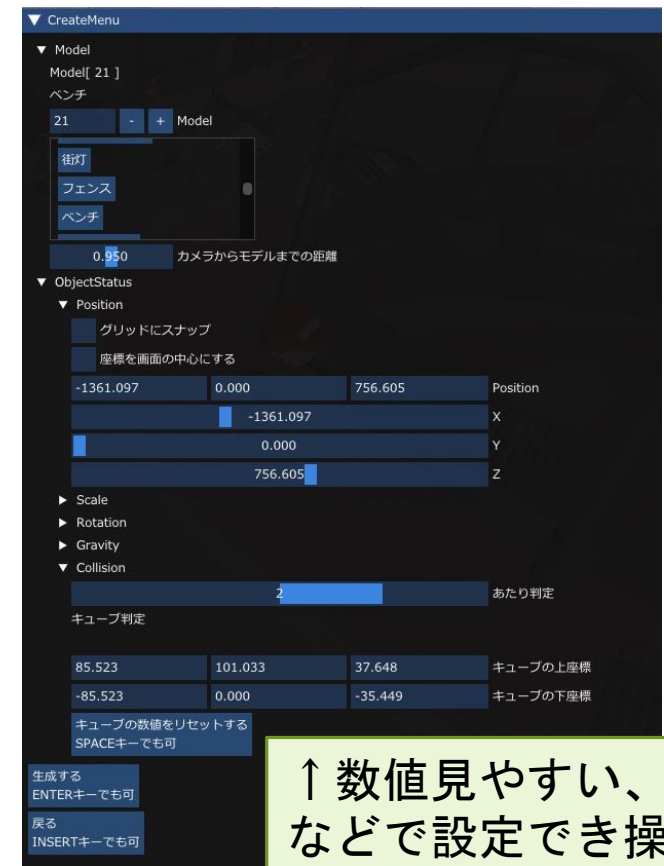
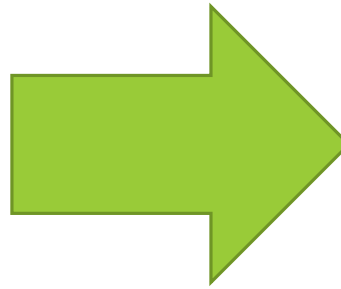


↑ 項目や設定中の数値が見づらい

オブジェクトを選択します。決定後マウスホイールかコントローラー左右で数字を選択
1にすると新しく生成します

↑ 8月に制作したもの

ImGuiを導入



↑ 数値見やすい、スライダー
などで設定でき操作しやすい

↑ 今回のもの

もともとは自前でUIを作っていたものの、試遊会などで多くの人に使ってもらった結果、「操作が複雑」「マウスでも操作できるようにしてほしい」といった意見が多くありました。はじめは、引き続き自前で作ろうと考えていましたが、自分で作るより効率が良い点や操作のしやすさ、編集項目の追加が簡単であるなどの理由から、ImGuiを導入することに決めました。その結果、作業効率・制作効率がかなり上がりました。

今後について

- より面白いゲームにするためにプログラムのリファクタリングやステージの追加、ステージ構成調整、ギミック追加などを行う。
- ツールを他の作品でも使用して、効率化とさらなる改善を図る
- Unityなどを参考にツールの操作性を上げる。
- カンマ区切りで見辛い.csvファイルから.jsonファイルなど管理しやすいファイル形式に変更する。

作品概要・ユカオトセ！



ジャンル：見下ろし型3Dアクション
制作環境：Visual Studio2019/C++ , DxLib
制作人数：1人
担当箇所：プログラム全般、UIデザイン
タイトル・リザルト画面画像作成

受賞歴

LABコン2023 商品スポンサー賞

LABコン2023 来場者投票数最多賞

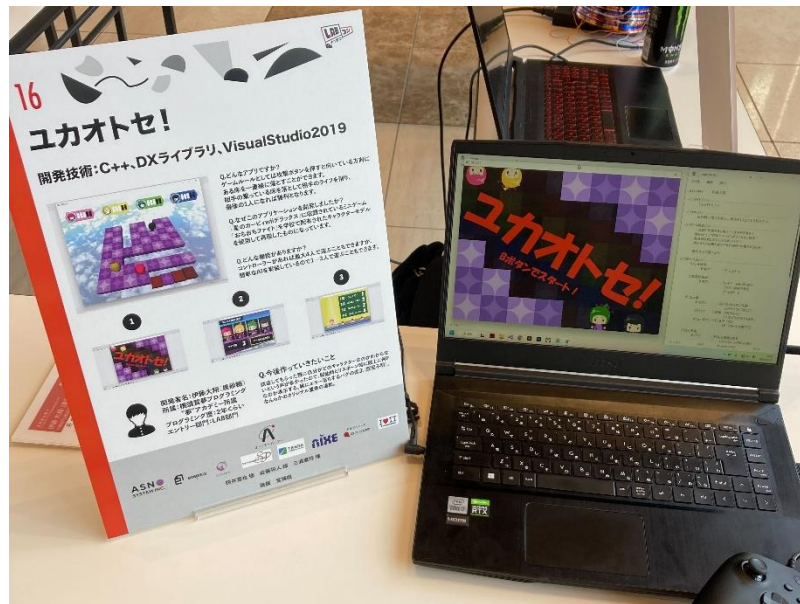
(詳細→ <https://techplay.jp/event/887201>)

ゲームクリエイター甲子園2023に展示

ゲーム概要

最大4人で対戦できるアクションゲームです。
プレイヤーは正面の床を攻撃し、相手ごと床を落とすことができます。
最後の1人まで生き残った人が優勝です。

コンテストについて



LABコンに出展した際の写真



ゲームクリエイター甲子園2023に展示した際の写真

私はもともと、地元・横須賀市の主催するプログラミング教室に参加しており、現在もそこで講師サポートのアルバイトをしています。その教室の業務委託先である株式会社イトナブの方からの誘いで、自分のプログラムスキルの向上のためにも仙台で行われたプログラミングコンテスト、LABコンに出展しました。

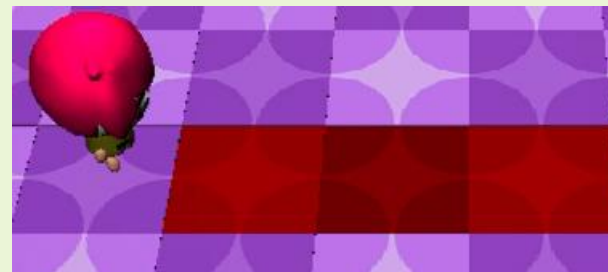
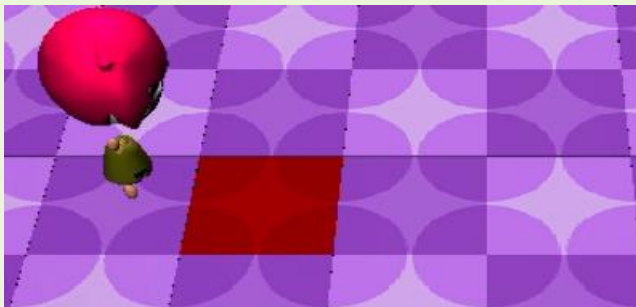
また、ゲームクリエイター甲子園にチーム作品が出展されることになった際に、主催のゲームクリエイターギルドの方のご厚意で、個人作品を展示する機会を頂いたので展示しました。

システム

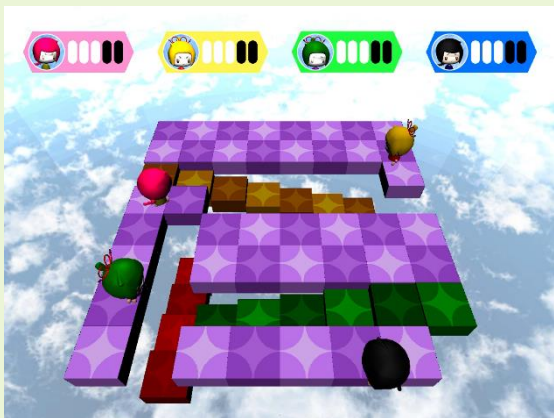


ゲーム開始前にAIとプレイヤーの切り替えをする画面を作り、切り替えられるようにしています。
AIは一定時間ごとにランダムな向きに切り替え、攻撃と移動を行います。
また、ライフも1～5の中で切り替えられるようにし、ライフ1でのデスマッチやライフ5でわいわい遊ぶなどの
選択肢を用意しました。

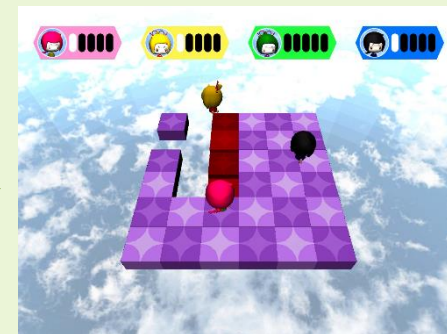
こだわり・床の色



↑ 順番に床の色が変わり、攻撃が迫ってくるハラハラ感を演出 ↑

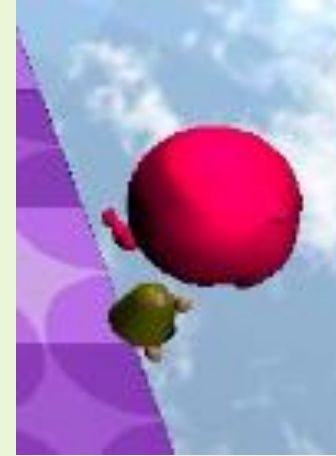
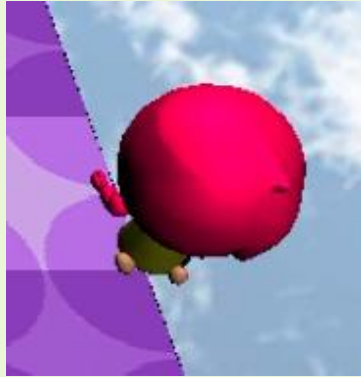


↑ 順番に落ちていく。
同時に複数方向から攻撃が来る



↑ ゲームオーバー者が出ると外周の色が
少しずつ変わり、フィールドが狭くなる

こだわり・落ちそう感+隙



↑交互に変化し、落ちそう感を表現↑
プレイヤーに落ちたくないと思わせる

また、画面端を向いている場合、攻撃ができない＝隙＝チャンス
であることをプレイヤーに教える

ご覧いただき
ありがとうございました
