# 情報メディア創成学類
# 情報メディア実験 A,B レポート

| | |
|---|---|
| 氏　　名 | 三田村 卓磨 |
| 学籍番号 | 202313640 |
| 提 出 日 | 西暦 2023　　年　　8 月　　9 日 |

| | |
|---|---|
| テーマ名 | 物理エンジンを使ったアプリケーション開発 |
| テーマ担当教員名 | 藤澤 誠 |
| 実施学期 | 春 ABC |

## 概 要


| |
|---|
| 本実験は、二つの主要なフェーズに分けて進行された。 |
| 前半は、物理エンジン Bullet の基本的な機能についての理解を深めた。 |
| 後半の部分では、具体的に Bullet を使用してゲームの開発を行った。 |
| 本レポートは、開発したゲームの概要に始まり、具体的な機能と |
| その実装方法、ゲームに対する考察、そして実験に対する感想など |
| 包括的な視点からの分析と評価に関するものである。 |

# 1. アプリケーションについて

## 1.1 アプリケーションの名前

このアプリケーションは、「Inkfluence:色魔法のパズル」というタイトルである。

## 1.2 アプリケーションの目的

このアプリケーションの目的は、物理エンジンが提供している機能を最大限に活用し、新たな視点から世界を探求することである。具体的には、現実世界における物理法則に対して、色が物体の性質に影響を及ぼすという特異な法則を加え、その結果として生じる世界がどれほど現実から逸脱し、また、どれほど面白くなるかを知りたかった。

## 1.3 アプリケーションの概要

このゲームは、物体の特性が色によって変化する世界を舞台としている。プレイヤーが操作する特別な筆は、色によって物体の特性を変えないという独特の能力を持ち、ギミックや障害物の色を変化させることで、仕掛けられたパズルを解きながら次々とステージを進むことができる。このゲームは 3D の 3 人称視点で構築されており、立体的な視野を提供しながら、プレイヤーにパズル解決の挑戦を投げかけるパズルアクションである。

# 2. 主要機能の仕様と実装

## 2.1 操作方法

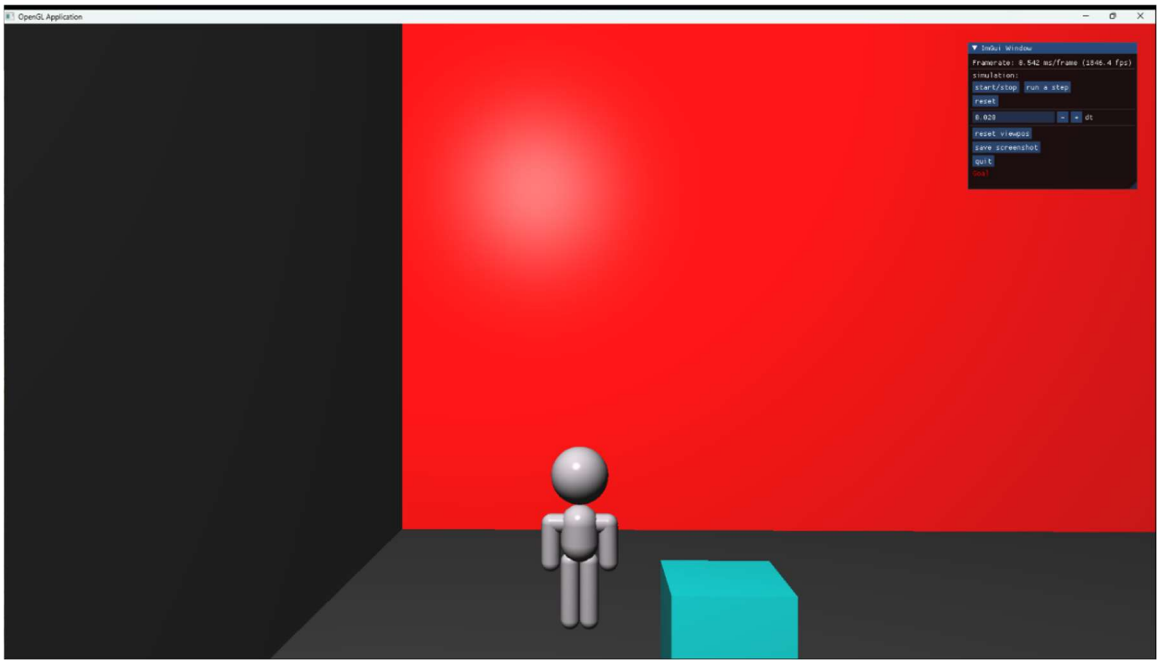


図 2-1 ジャンプしている様子

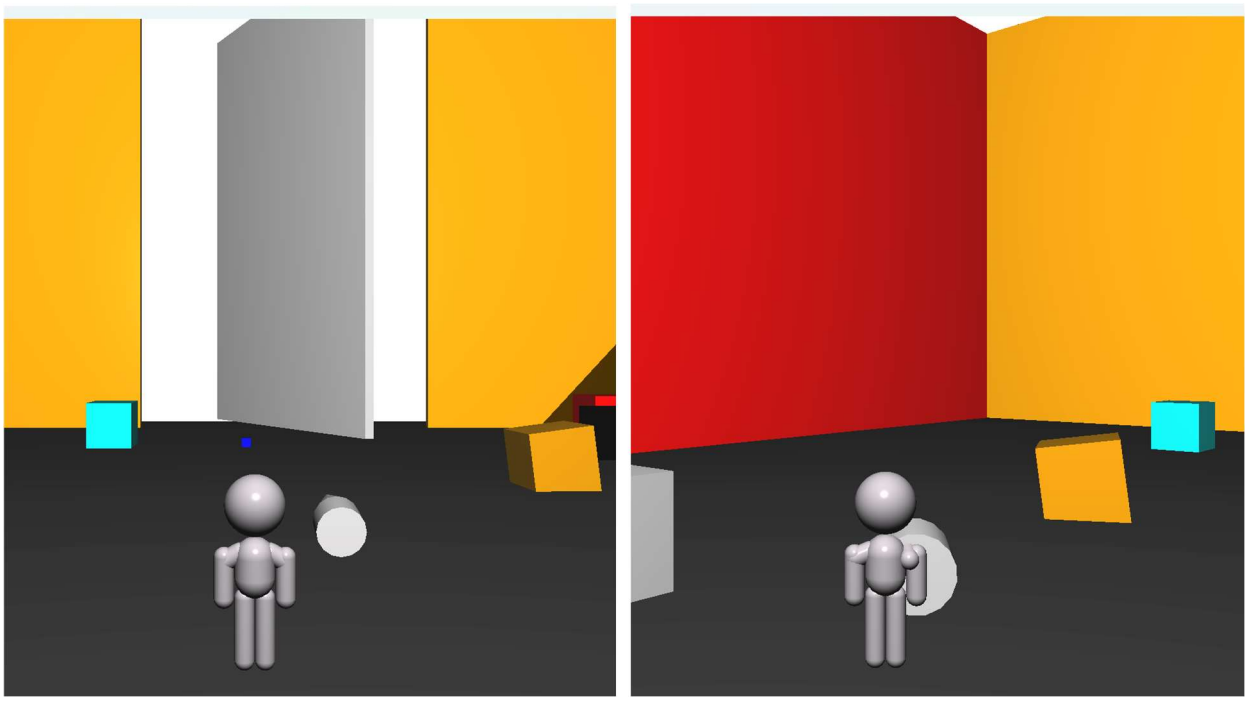

図 2-2 視線を回転させている様子

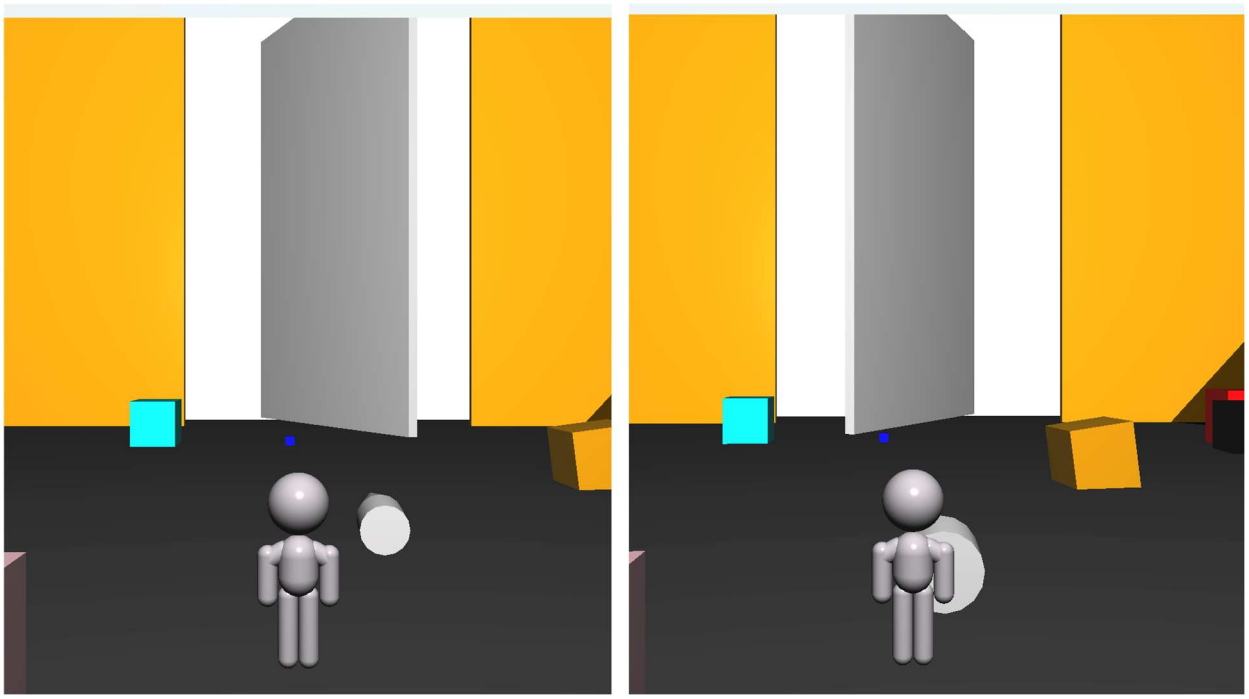

図 2-3 筆を持つ様子

### 2.1.1　操作方法の説明

キーボードの W キーで前に、A キーで左に、S キーで後ろに、D キーで右に進むことができる。また、(図 2-1)にあるように、Space キーを押すことでジャンプが可能である。視点の操作に関しては、マウスを x 方向に動かすことで、動かした量に合わせて(図 2-2)のように視点を回転させることが可能である。加えて、プレイヤーの近くに筆が存在する場合、(図 2-3)のようにマウスの左クリックで筆を持つ操作が可能になる。

### 2.1.2　操作の実現

以下の(Code1)によって実装した。

このコードは 4 つの部分に分けられる。まず、Keyboard 関数では、各キーの押下と放出に応じて、特定の操作フラグを制御し、方向移動を実行している。また、btRigidBody->applyCentralForce 関数をプレイヤーの各部位で行うことでジャンプを実現している。次に、Timer 関数では、プレイヤーの移動を制御し、w, a, s, d キーに応じたローカル座標の方向に力を加えるロジックが実装されている。ここでは、LocalToWorldTrans 関数によってローカル座標系からワールド座標系への変換が行われ、移動方向が計算される。mouse_callback 関数では、マウスの瞬間移動量に基づいて btQuaternion に回転量をかけることで、プレイヤーの回転を制御するロジックが記述されている。最後に、Mouse 関数では、左ボタンが押された場合には、GetClosestObj 関数によってプレイヤーから最も近いオブジェクトの選択が、Mouse イベント関数内での btGeneric6DofConstraint の生成によって接続が行われ、左ボタンが放された場合には、btGeneric6DofConstraint の消去によってその接続が解除される。

こだわりポイントは 2 つある。1 つは、Player を 7 つの Constraint でつないだ 8 つの剛体の集合として実装することで、関節で各部位がつながっている人間の動きを模倣したことである。もう 1 つは、イベント関数の Keyboard 関数では操作フラグのみを制御し、力の計算は一定時間ごとに実行される Timer 関数で行うことで、安定したプレイヤーの操作を実現したことである。

```
btScalar max_speed = 30.0;
btScalar force_coef = 100;
btScalar lr_force_coef = 0.7;
btVector3 jump_impulse = btVector3(0, 45, 0);
bool w_flag = false, a_flag = false, s_flag = false, d_flag = false, release_flag = false;
extern btRigidBody* player_rigid[8];

void Keyboard(GLFWwindow* window, int key, int scancode, int action, int mods)
{
	case GLFW_KEY_SPACE: // ジャンプ
	   player_rigid[0]->applyCentralImpulse(jump_impulse);
	   for(int i=1;i<8;i++)
	      player_rigid[i] ->applyCentralImpulse(jump_impulse/20);
	   break;
	case GLFW_KEY_W: // 前進
	   w_flag = true;
	   break;
	case GLFW_KEY_A: // 左移動
```

```
            a_flag = true;
            break;
        case GLFW_KEY_S: // 後退
            s_flag = true;
            break;
        case GLFW_KEY_D: // 右移動
            d_flag = true;
            break;
        default:
            break;
        }
    }
    else if (action == GLFW_RELEASE)
    {
        release_flag = true;
        switch (key)
        {
        case GLFW_KEY_W: // 前進
            w_flag = false;
            break;
        case GLFW_KEY_A: // 左移動
            a_flag = false;
            break;
        case GLFW_KEY_S: // 後退
            s_flag = false;
            break;
        case GLFW_KEY_D: // 右移動
            d_flag = false;
            break;
        default:
            break;
        }
    }
}


btVector3 LocalToWorldTrans(btRigidBody* local_body, btVector3 local_vector)
{
    btTransform trans = local_body->getWorldTransform();
    btQuaternion rot = trans.getRotation().normalize();
    btVector3 world_vector = quatRotate(rot, local_vector);
    return world_vector;
}


void Timer(void)
{
    if (g_animation_on) {
        if (g_dynamicsworld) {
            MoveCameraWithRigid(g_view, player_rigid[0]); //カメラを後ろに配置
        }
        //移動処理
        if (w_flag || a_flag || s_flag || d_flag)
        {
            btVector3 force = btVector3(0, 0, 0);
```

```
            btVector3 front_back_force = LocalToWorldTrans(player_rigid[0], btVector3(0, 0,
1)).normalize();
            btVector3 left_right_force = LocalToWorldTrans(player_rigid[0], btVector3(1, 0,
0)).normalize()*lr_force_coef;
            if (w_flag)
                force -= front_back_force;
            if (s_flag)
                force += front_back_force;
            if (a_flag)
                force -= left_right_force;
            if (d_flag)
                force += left_right_force;
            if (force.length() != 0)
            {
                if (force.length() > front_back_force.length())
                    force.normalize();
                player_rigid[0]->applyCentralForce(force * force_coef);
                for(int i=1;i<8;i++)
                    player_rigid[i]->applyCentralForce(force * force_coef/20);
            }
        }
}

double lastX = 0;
bool firstMouse = true;
void mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
    //可変パラメータ 1 で 1cm の移動につき 100 度回転
    double rotatespeed = 1.0;
    if (firstMouse) // 初めてのマウス入力の場合
    {
        lastX = xpos;
        firstMouse = false;
    }

    // マウスの移動量を計算
    double xoffset = lastX - xpos;
    lastX = xpos;

    // マウスの移動量に基づいてキャラクターを回転
    if (player_rigid[0]) {
        btTransform trans;
        trans.setIdentity();
        for (int i = 0; i < 8; i++)
        {
            player_rigid[i]->getMotionState()->getWorldTransform(trans);
            btQuaternion rotation;
            if(i==4||i==5)
                rotation = btQuaternion(btVector3(1, 0, 0), btScalar(xoffset * rotatespeed * RX_PI /
180.0));
            else
                rotation = btQuaternion(btVector3(0, 1, 0), btScalar(xoffset * rotatespeed * RX_PI /
180.0));
            // 現在の姿勢と新たな回転を合成
            btQuaternion newRotation = trans.getRotation() * rotation;
            trans.setRotation(newRotation);
```

```
            // 新たな姿勢を設定
            player_rigid[i]->setWorldTransform(trans);
        }
    }
}

//近くの筆(index=98)を見つける
btRigidBody* GetClosestObj(float maxRad) {
    const btCollisionObjectArray& objArray = g_dynamicsworld->getCollisionObjectArray();
    btRigidBody* closestObject = nullptr;
    float minDist = maxRad;

    for (int i = 0; i < objArray.size(); ++i) {
        btRigidBody* body = btRigidBody::upcast(objArray[i]);
        if (body&&body->getUserIndex() == SPECIAL_INDEX) {
            btVector3 diff = player_rigid[0]->getWorldTransform().getOrigin() - body-
>getWorldTransform().getOrigin();
            float dist = diff.length();
            if (dist < minDist) {
                minDist = dist;
                closestObject = body;
            }
        }
    }
    return minDist <= maxRad ? closestObject : nullptr;
}

void Mouse(GLFWwindow* window, int button, int action, int mods)
{
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS)
    {
        left_pressing = true;
        //プレイヤーから最も近いオブジェクトを探します
        btRigidBody* pickedObject = GetClosestObj(5.0);
        if (pickedObject == nullptr)
        {
            std::cout << "nothing¥n";
        }
        if (pickedObject)
        {
            btQuaternion a(0, 0, 0, 1), b(0, 0, 0, 1);
            if (!player_rigid[0])
            {
                cout << "null playerRigid" << endl;
                return;
            }

            btTransform frame_in_a(a, btVector3(0,0, -0.1)), frame_in_b(b, btVector3(0, 0, 0.6));
            //A のほうが重いので A=true ワールド座標での指定を行うこと
            btGeneric6DofConstraint* player2pencilcon = new btGeneric6DofConstraint(*player_rigid[7],
*pickedObject, frame_in_a, frame_in_b, true);
            // 全ての軸で移動を無効化
            for (int i = 0; i < 3; i++) {
                player2pencilcon->setLimit(i, 0, 0);
            }
            player2pencilcon->setAngularLowerLimit(btVector3(0,0, -RX_PI / 12));
```

```
            player2pencilcon->setAngularUpperLimit(btVector3(0, 0,RX_PI / 12));
            // 制約を追加
            g_dynamicsworld->addConstraint(player2pencilcon, true);
            pickConstraint = player2pencilcon;
        }
    }
    else if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_RELEASE)
    {
        left_pressing = false;
        // マウスのボタンが離されたときに、"ピック"操作を終了
        if (pickConstraint)
        {
            g_dynamicsworld->removeConstraint(pickConstraint);
            delete pickConstraint;
            pickConstraint = nullptr;
        }
    }
}
```

Code1 プレイヤー操作のプログラム

## 2.2 ステージ遷移

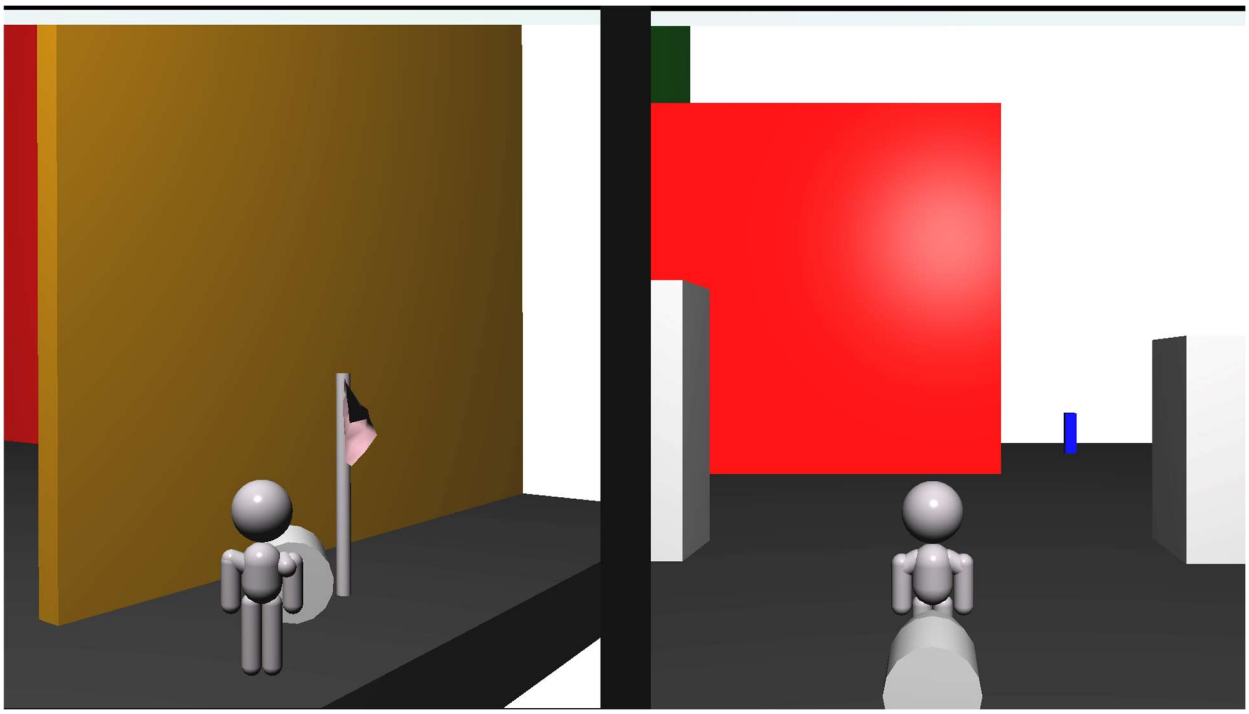

図 2-4 ステージ遷移の様子

### 2.2.1　説明

(図 2-4)に示すように筆がクリアを示す flag と衝突したとき、次のステージの開始地点にリスポーンする。

### 2.2.2　ステージ遷移の実装

以下の(Code2)によって実装した。

このコードは clear_stage と current_stage 変数によって、現在のステージとクリアしたステージの情報を管理している。MyContactResultCallback クラス内で特定の条件に一致する場合、ステージをクリアし次のステージへ進む処理が行われる。クリアしたステージに応じた処理が main 関数内で分岐され、ステージのクリーンアップと初期化、次のステージの生成が行われる。こだわりポイントとしては、ステージ遷移の流れが明確にコード化され、拡張性を考慮した設計となっている点である。

```
int clear_stage = 0;
int current_stage = 1;

//筆(index=98)が何かと接触したときに起動するイベント関数
class MyContactResultCallback : public btCollisionWorld::ContactResultCallback
{
…
        //checkpoint+ステージ破壊と作成のフラッグ=index96
        else if (user_index == 96)
        {
            int index2 = n_OtherBody->getUserIndex2();
            //index2はフラッグによって異なる
            switch (index2)
            {
            case 1:
                clear_stage = 1;
                current_stage = 2;
                break;
            case 2:
                clear_stage = 2;
                current_stage = 3;
                break;
            case 3:
                clear_stage = 3;
```

```
                break;
            default:
                cout << "Undefined stage number." << endl;
                break;
            }
        }
    return 0;
...
    }
};

int main(int argc, char* argv[])
{
    if (clear_stage!=0)
    {
        switch (clear_stage)
        {
        case 1:
            CleanBullet();
            InitBullet();
            MakeStage2();
            break;
        case 2:
            CleanBullet();
            InitBullet();
            move_rigid = nullptr;
            MakeStage3();
            break;
        case 3:
            cout << "Congrats." << endl;
        default:
            break;
        }
        clear_stage = 0;
    }
}
```

Code2 ステージ遷移のプログラム

## 2.3 筆、壁、床に特殊な役割を持たせる機能

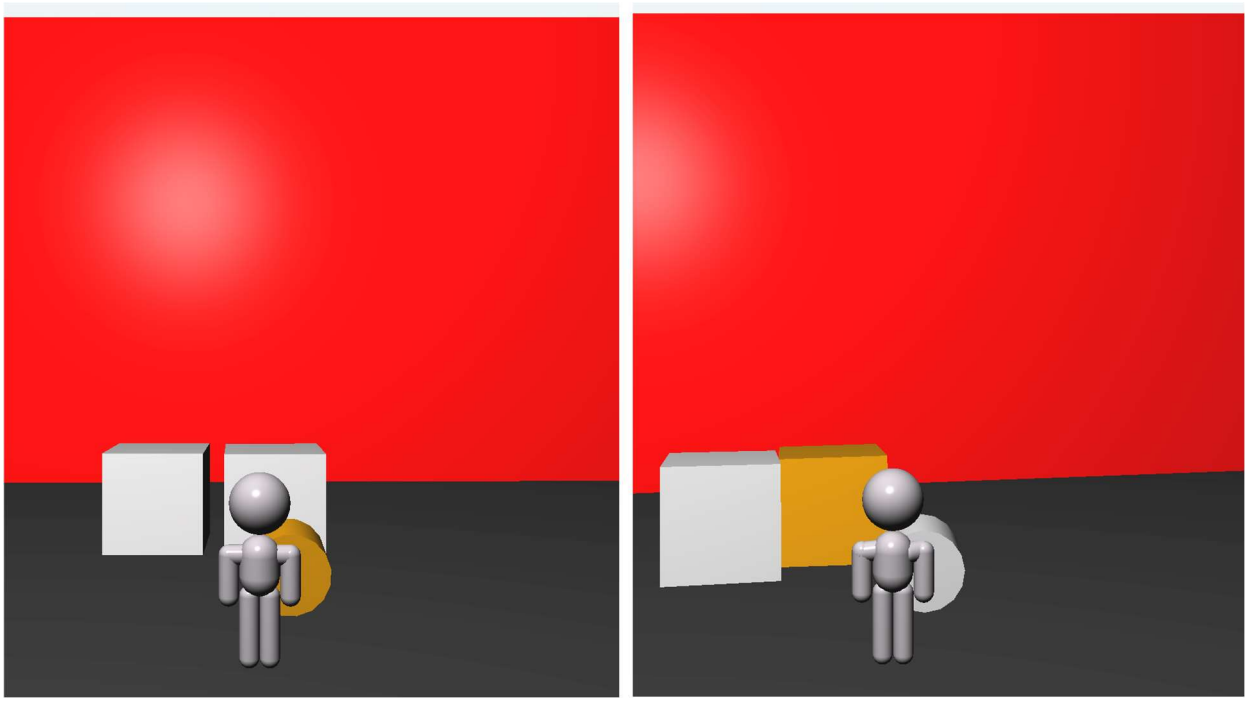

図 2-5 筆から障害物に色を移す様子

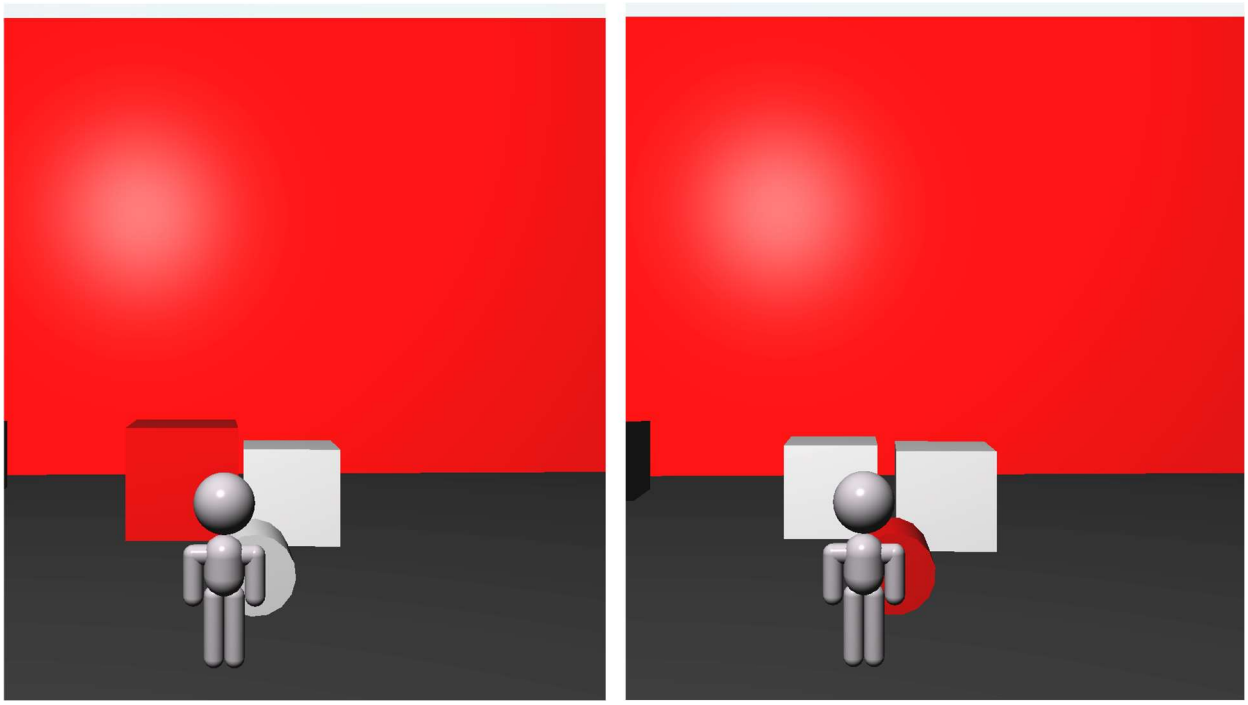

図 2-6 障害物から色をもらう様子

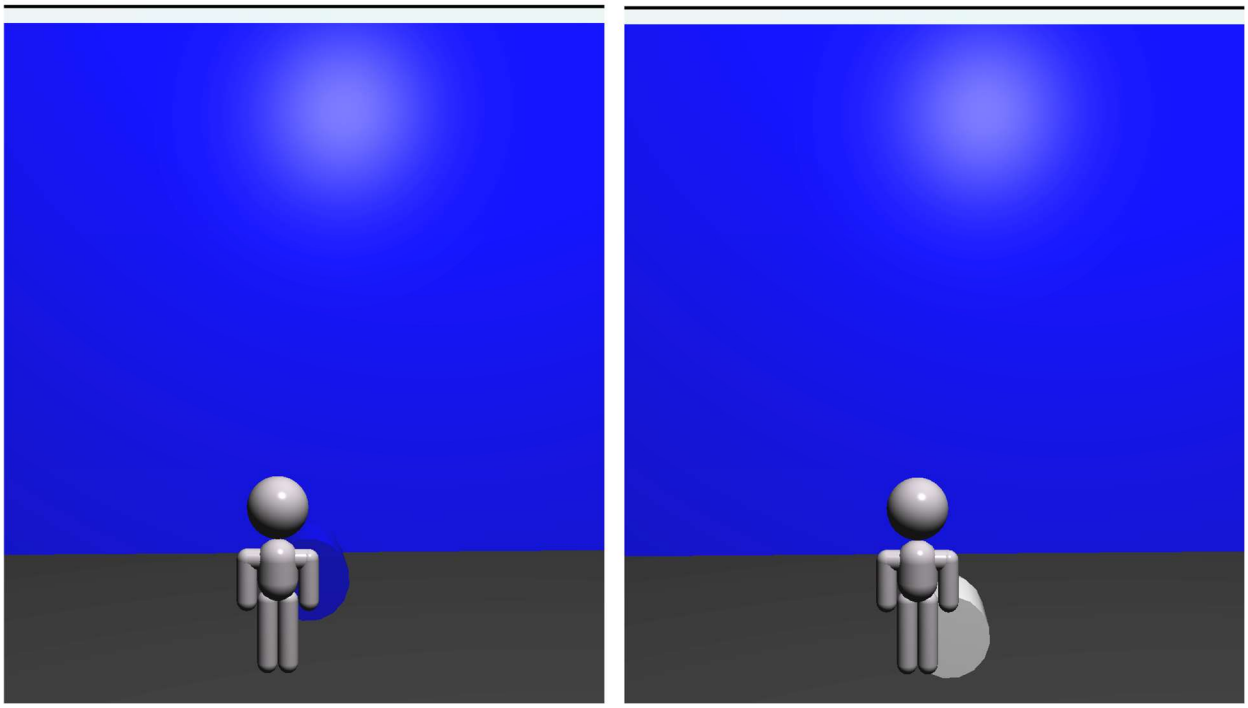

図 2-7 筆から床に色を移す様子

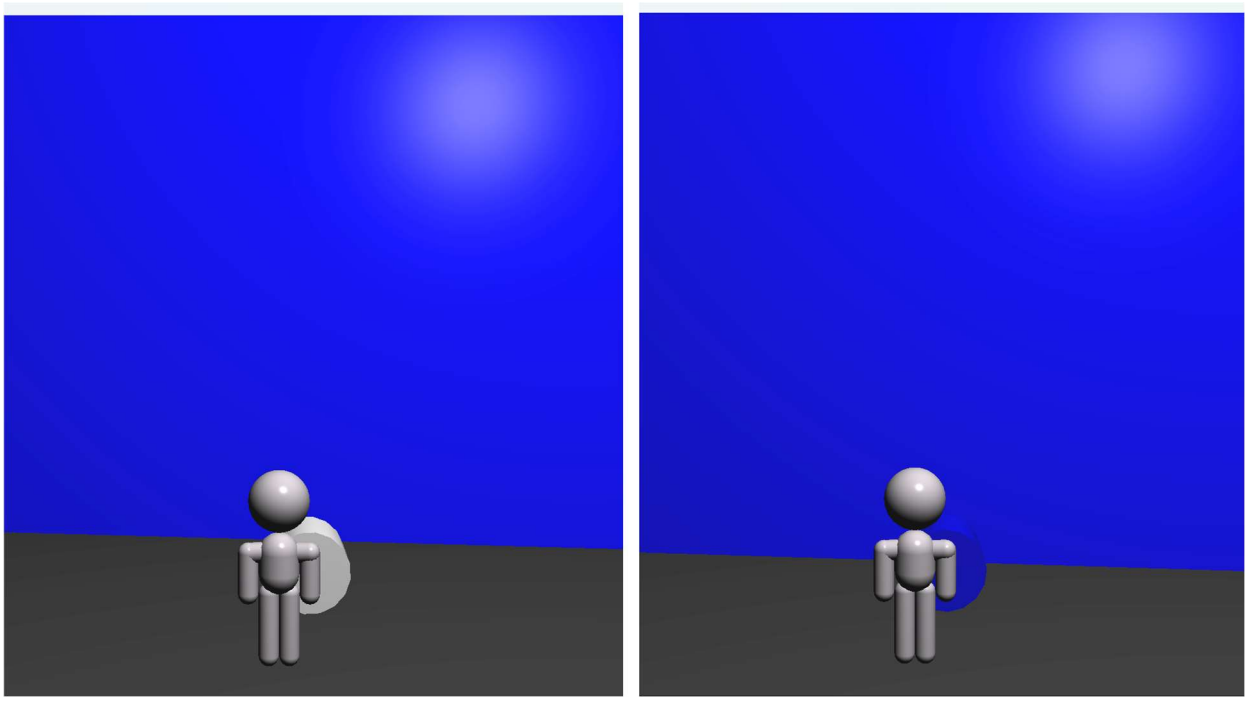

図 2-8 壁から色をもらう様子

### 2.3.1 説明

このゲームには特殊な 3 種類のオブジェクトが存在する。

筆は色をオブジェクトから他のオブジェクトに移動させることができる役割を持ち、色によって性質が変化しない。(図 2-5)に示すように障害物に色を移したり、(図 2-6)に示すように障害物から

色をもらうことができる。

床は、(図 2-7)に示すように筆から色をもらっても性質が変化しないオブジェクトであり、水性絵の具に対する水のような役割を持つ。

壁は、(図 2-8)に示すように筆に特定の色を無限に供給するオブジェクトであり、絵の具のような役割を持つ。ただし、壁が供給する色は 1 種類である。

### 2.3.2 機能の実装

以下の(Code3)によって実装した。

このコードは、主に 2 つの部分から構成されている。1 つは、オブジェクトの描画を行う DrawBulletObjects 関数がある。この関数内では、さまざまな色の定義と、2 種類の userindex に応じて描画を行うロジックが実装されている。もう 1 つは、MyContactResultCallback クラスである。これは、筆が衝突するたびに呼ばれるコールバックで衝突したオブジェクトの情報に応じて色(userindex)や性質の変更を実施する。障害物の性質の変更については(2.4)で解説する。

こだわりポイントは、多彩な色のオブジェクトを描画するための色設定と、userindex によってオブジェクト間の相互作用を高度にコントロールしている点である。

```
//描画関数
void DrawBulletObjects(void* x = 0)
{
…
   static const GLfloat main_white[] = { 0.808, 0.78, 0.808, 1.0 };   // メインオブジェクト:(0)
   static const GLfloat black[] = { 0.0, 0.0, 0.0, 1.0 };   // 黒(1)
   static const GLfloat white[] = { 1.0, 1.0, 1.0, 1.0 };   // 白(2)
   static const GLfloat cyan[] = { 0.0, 1.0, 1.0, 1.0 };// (3)
   static const GLfloat clear_yellow[] = { 1.0, 1.0, 0.0, 1.0 };   //(4)
   static const GLfloat deep_green[] = { 0.0, 0.5, 0.0, 1.0 }; // (5)
   static const GLfloat pink[] = { 1.0, 0.75,0.8, 1.0 };// (6)  弾性体
   static const GLfloat red[] = { 1.0, 0.0, 0.0, 1.0 }; // (7)  大きさ↑
   static const GLfloat blue[] = { 0.0, 0.0, 1.0, 1.0 };// (8)  大きさ↓
   static const GLfloat gray[] = { 0.5, 0.5, 0.5, 1.0 };// (9)  慣性モーメント+
   static const GLfloat orange[] = { 1.0, 0.65, 0.0, 1.0 };// (10)  慣性モーメント-
   static const GLfloat shine_black[] = { 0.2,0.2,0.2 }; //index=97

…
   if (g_dynamicsworld) {
         …
         if (shapetype == SOFTBODY_SHAPE_PROXYTYPE) {
       …
         }
         else {
         …
            if (body) {
               switch (body->getUserIndex())
               {
               //壁(色源)または筆
               case 98:
               case 99:
                  switch (body->getUserIndex2()) {
                  case 2:
```

```
                        glMaterialfv(GL_FRONT, GL_DIFFUSE, white);
                        break;
                    case 3:
                        glMaterialfv(GL_FRONT, GL_DIFFUSE, cyan);
                        break;
                    case 4:
                        glMaterialfv(GL_FRONT, GL_DIFFUSE, clear_yellow);
                        break;
                    case 5:
                        glMaterialfv(GL_FRONT, GL_DIFFUSE, deep_green);
                        break;
                    case 6:
                        glMaterialfv(GL_FRONT, GL_DIFFUSE, pink);
                        break;
                    case 7:
                        glMaterialfv(GL_FRONT, GL_DIFFUSE, red);
                        break;
                    case 8:
                        glMaterialfv(GL_FRONT, GL_DIFFUSE, blue);
                        break;
                    case 9:
                        glMaterialfv(GL_FRONT, GL_DIFFUSE, gray);
                        break;
                    case 10:
                        glMaterialfv(GL_FRONT, GL_DIFFUSE, orange);
                        break;
                    case 97:
                        glMaterialfv(GL_FRONT, GL_DIFFUSE, shine_black);
                        break;
                    }
                    break;
                //床の場合
                 case 97:
                    glMaterialfv(GL_FRONT, GL_DIFFUSE, shine_black);
                    break;
                 default:
                    glMaterialfv(GL_FRONT, GL_DIFFUSE, black);
                    break;
                 }
             }
...
}

//筆(index=98)が衝突するたびに呼ばれるコールバッククラス
class MyContactResultCallback : public btCollisionWorld::ContactResultCallback
{
public:
    bool collision = false;

    virtual btScalar addSingleResult(btManifoldPoint& cp,
        const btCollisionObjectWrapper* colObj0Wrap, int partId0, int index0,
        const btCollisionObjectWrapper* colObj1Wrap, int partId1, int index1)
    {
        // Get the rigidbodies involved in the collision
        const btRigidBody* body0 = btRigidBody::upcast(colObj0Wrap->getCollisionObject());
        const btRigidBody* body1 = btRigidBody::upcast(colObj1Wrap->getCollisionObject());
        collision = true;
```

```
        //special が筆を指している
        const btRigidBody* specialBody = (body0->getUserIndex() == SPECIAL_INDEX) ? body0 : body1;
       //筆と接触した剛体
        const btRigidBody* otherBody = (body0->getUserIndex() == SPECIAL_INDEX) ? body1 : body0;
        time_t temp_t = clock() - touch_time;
        //一定時間が経過+触った先が変更可能(黒またはプレイヤーでない)場合
        int user_index = otherBody->getUserIndex();
        if (user_index != 0 && user_index != 1 && (temp_t > 1000))
        {
           // Remove const modifier
           btRigidBody* n_OtherBody = const_cast<btRigidBody*>(otherBody);
           btRigidBody* n_SpecialBody = const_cast<btRigidBody*>(specialBody);

           //触った先が色源(壁)の場合
           if (user_index == 99)
           {
              if (otherBody->getUserIndex2() == 97)
                 n_SpecialBody->setUserIndex2(2);
              else
                 n_SpecialBody->setUserIndex2(n_OtherBody->getUserIndex2());
           }
          //触った先が床の場合
           else if (user_index == 97)
           {
              n_SpecialBody->setUserIndex2(2);
           }
           …
           // 筆の Index2 が 2(色吸収モード)のとき
           else if (n_SpecialBody->getUserIndex2() == 2)
           {
              …
              n_SpecialBody->setUserIndex2(n_OtherBody->getUserIndex());
              n_OtherBody->setUserIndex(2);
           }
           //筆の Index2 が塗りモードの場合(else if で競合回避)
           else if (3 <= n_SpecialBody->getUserIndex2() && n_SpecialBody->getUserIndex2() <= 10)
           {
              if (user_index == 2)
              {
                 …
                 n_OtherBody->setUserIndex(n_SpecialBody->getUserIndex2());
                 n_SpecialBody->setUserIndex2(2);
              }
              //一回白に戻す必要性がある(性質の重ね塗り対策)
              else
              {

                 …
                 n_OtherBody->setUserIndex(n_SpecialBody->getUserIndex2());
                 n_SpecialBody->setUserIndex2(2);
              }
           }
           else
           {
              cout << "Unkown Color Detected." << endl;
           }
           touch_time = clock();
```

```
        }
        return 0;
    }
};
```

Code3 筆、壁、床の持つ役割を実装するプログラム

## 2.4 色による障害物の物理特性変化

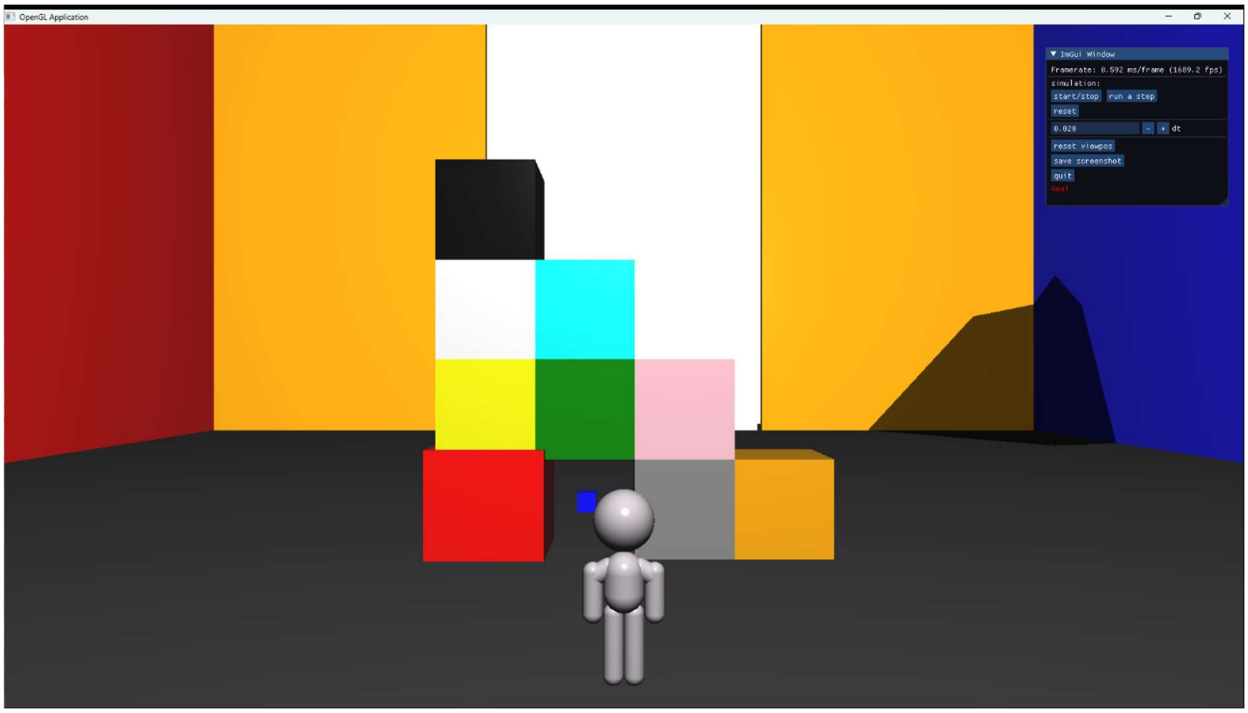


図 2-9 実装された色の一覧

### 2.4.1 特性の説明

このゲームには(図 2-9)に示すように、未実装のピンク色を除いた 9 色の特性変化/性質が存在する。1 つ目は黒色(index=1)であり、この色の障害物は物理特性変化を持たず、また 9 色の中で唯一他の色で塗り替えることができない。2 つ目は白色(index=2)であり、この色の障害物は物理特性変化を持たない。3 つ目は水色(index=3)であり、この色の障害物は摩擦係数が 1/50 になる。4 つ目は黄色(index=4)であり、この色の障害物は質量が 1/100 になる。5 つ目は深緑色(index=5)であり、この色の障害物は反発係数が 12 倍になる。6 つ目は赤色(index=7)であり、この色の障害物は大きさが 1.2 倍になる。7 つ目は青色(index=8)であり、この色の障害物は質量が 1/5 になる。8 つ目は灰色(index=9)であり、この色の障害物は慣性モーメントが 10,000 倍になる。9 つ目はオレンジ色(index=10)であり、この色の障害物は慣性モーメントが 1/100 になる。

### 2.4.2　実装

筆が何かと衝突したときに呼ばれるクラス MyContactResultCallback を定義し、その中に障害物との衝突時の処理を記述している。この動作は(code5)で実装されている。

物理特性の変更については、摩擦係数を変化させるために btRigidBody->setFriction 関数、質量または慣性モーメントを変化させるための btRigidBody->setMassProps 関数、反発係数を変化させるための btRigidBody->setRestitution 関数を使用している。また、元の btRigidBody の情報を btRigidBody->getLinearVelocity 関数、btRigidBody->getWorldTransform 関数などで取得し、それに btRigidBody->setLocalScaling 関数で変更した情報を加えて RigidBody を再作成することで大きさの変更を実装している。さらに、物理特性の変更に関する係数はグローバル変数としてコード内の一箇所にまとめられている。この動作は(Code4)で実装されている。こだわりのポイントは 2 つある。1 つは、2 種類の userindex と switch 文を駆使することでオブジェクト間の相互作用を高度にコントロールしている点である。もう一つは、障害物が白または黒以外の色から異なる色に塗り替えられるときに一旦白色を経由して塗り替えることで、2 種類以上の物理特性変化を持つオブジェクトが作成されるのを防ぎ、ステージのデバッグをやりやすくしている点である。この工夫によって、考えられる解法は最大でも 8^x(x:=オブジェクトの数)通りまで絞られる。

```
btScalar fric_coef = 0.02;
btScalar mass_coef = 0.01;
btScalar restitution_coef = 12;
btScalar expand_coef = 1.2;
btScalar reduce_coef = 0.2;
btScalar incr_inertia_coef = 10000;
btScalar decr_inertia_coef = 0.01;

btRigidBody* SwitchColorChange(btRigidBody* n_SpecialBody, btRigidBody* n_OtherBody)
{
   btScalar b_mass;
   if (n_OtherBody->getInvMass())
      b_mass=1 / n_OtherBody->getInvMass();
   else
      b_mass = 0;
   btVector3 b_inertia = n_OtherBody->getLocalInertia();
   btVector3 b_scaling = n_OtherBody->getCollisionShape()->getLocalScaling();
   int o_index;
   btCollisionShape* o_shape{};
   btTransform o_trans;
   btMotionState* motion_state_a{};
   btRigidBody* delete_body = n_OtherBody;
   btRigidBody* return_body{};
   btVector3 b_lin_velo;
   btVector3 b_ang_velo;
   btScalar o_friction;
   btScalar o_restitution;

   switch (n_SpecialBody->getUserIndex2()) {
   case 3:
      n_OtherBody->setFriction(n_OtherBody->getFriction() * fric_coef);
      return_body = n_OtherBody;
```

```
        break;
    case 4:
        n_OtherBody->setMassProps(b_mass * mass_coef, b_inertia);
        return_body = n_OtherBody;
        break;
    case 5:
        n_OtherBody->setRestitution(n_OtherBody->getRestitution() * restitution_coef);
        return_body = n_OtherBody;
        break;
    case 6:
        break;
    case 7:
        b_lin_velo = n_OtherBody->getLinearVelocity();
        b_ang_velo = n_OtherBody->getAngularVelocity();
        o_index = n_OtherBody->getUserIndex();
        o_trans = n_OtherBody->getWorldTransform();
        o_shape = n_OtherBody->getCollisionShape();
        o_friction = n_OtherBody->getFriction();
        o_restitution = n_OtherBody->getRestitution();
        o_shape->setLocalScaling(b_scaling * expand_coef);
        g_dynamicsworld->removeRigidBody(n_OtherBody);
        motion_state_a = n_OtherBody->getMotionState();
        delete motion_state_a;
        delete delete_body;
        return_body=CreateRigidBody(b_mass,o_trans, o_shape,
g_dynamicsworld,&o_index,RX_COL_OBST,RX_COL_ALL,false,b_inertia);
        return_body->setLinearVelocity(b_lin_velo);
        return_body->setAngularVelocity(b_ang_velo);
        return_body->setFriction(o_friction);
        return_body->setRestitution(o_restitution);
        break;
    case 8:
        b_lin_velo = n_OtherBody->getLinearVelocity();
        b_ang_velo = n_OtherBody->getAngularVelocity();
        o_index = n_OtherBody->getUserIndex();
        o_trans = n_OtherBody->getWorldTransform();
        o_shape = n_OtherBody->getCollisionShape();
        o_friction = n_OtherBody->getFriction();
        o_restitution = n_OtherBody->getRestitution();
        o_shape->setLocalScaling(b_scaling * reduce_coef);
        g_dynamicsworld->removeRigidBody(n_OtherBody);
        motion_state_a = n_OtherBody->getMotionState();
        delete motion_state_a;
        delete delete_body;
        return_body = CreateRigidBody(b_mass, o_trans, o_shape, g_dynamicsworld,&o_index, RX_COL_OBST,
RX_COL_ALL, false, b_inertia);
        return_body->setLinearVelocity(b_lin_velo);
        return_body->setAngularVelocity(b_ang_velo);
        return_body->setFriction(o_friction);
        return_body->setRestitution(o_restitution);
        break;
    case 9:
        n_OtherBody->setMassProps(b_mass, b_inertia * incr_inertia_coef);
        return_body = n_OtherBody;
        break;
    case 10:
        n_OtherBody->setMassProps(b_mass, b_inertia * decr_inertia_coef);
```

```
        return_body = n_OtherBody;
        break;
    default:
        cout << "error¥n";
        break;
    }
    return return_body;
}


btRigidBody* RigidMakedWhite(btRigidBody* n_SpecialBody, btRigidBody* n_OtherBody)
{
    btScalar b_mass;
    if (n_OtherBody->getInvMass())
        b_mass = 1 / n_OtherBody->getInvMass();
    else
        b_mass = 0;
    btVector3 b_inertia = n_OtherBody->getLocalInertia();
    btVector3 b_scaling = n_OtherBody->getCollisionShape()->getLocalScaling();
    int o_index;
    btCollisionShape* o_shape{};
    btTransform o_trans;
    btMotionState* motion_state_a{};
    btRigidBody* delete_body = n_OtherBody;
    btRigidBody* return_body{};
    btVector3 b_lin_velo;
    btVector3 b_ang_velo;
    btScalar o_friction;
    btScalar o_restitution;

    switch (n_OtherBody->getUserIndex()) {
        //2は何も起きない 2,2の特殊パターン
    case 2:
        return_body = n_OtherBody;
        break;
    case 3:
        n_OtherBody->setFriction(n_OtherBody->getFriction() / fric_coef);
        return_body = n_OtherBody;
        break;
    case 4:
        n_OtherBody->setMassProps(b_mass / mass_coef, b_inertia);
        return_body = n_OtherBody;
        break;
    case 5:
        n_OtherBody->setRestitution(n_OtherBody->getRestitution() / restitution_coef);
        return_body = n_OtherBody;
        break;
    case 6:
        return_body = n_OtherBody;
        break;
    case 7:
        b_lin_velo = n_OtherBody->getLinearVelocity();
        b_ang_velo = n_OtherBody->getAngularVelocity();
        o_index = n_OtherBody->getUserIndex();
        o_trans = n_OtherBody->getWorldTransform();
        o_shape = n_OtherBody->getCollisionShape();
        o_friction = n_OtherBody->getFriction();
```

```
        o_restitution = n_OtherBody->getRestitution();
        o_shape->setLocalScaling(b_scaling / expand_coef);
        g_dynamicsworld->removeRigidBody(n_OtherBody);
        motion_state_a = n_OtherBody->getMotionState();
        delete motion_state_a;
        delete delete_body;
        return_body = CreateRigidBody(b_mass, o_trans, o_shape, g_dynamicsworld, &o_index,
RX_COL_OBST, RX_COL_ALL, false, b_inertia);
        return_body->setLinearVelocity(b_lin_velo);
        return_body->setAngularVelocity(b_ang_velo);
        return_body->setFriction(o_friction);
        return_body->setRestitution(o_restitution);
        break;
    case 8:
        b_lin_velo = n_OtherBody->getLinearVelocity();
        b_ang_velo = n_OtherBody->getAngularVelocity();
        o_index = n_OtherBody->getUserIndex();
        o_trans = n_OtherBody->getWorldTransform();
        o_shape = n_OtherBody->getCollisionShape();
        o_friction = n_OtherBody->getFriction();
        o_restitution = n_OtherBody->getRestitution();
        o_shape->setLocalScaling(b_scaling / reduce_coef);
        g_dynamicsworld->removeRigidBody(n_OtherBody);
        motion_state_a = n_OtherBody->getMotionState();
        delete motion_state_a;
        delete delete_body;
        return_body = CreateRigidBody(b_mass, o_trans, o_shape, g_dynamicsworld, &o_index,
RX_COL_OBST, RX_COL_ALL, false, b_inertia);
        return_body->setLinearVelocity(b_lin_velo);
        return_body->setAngularVelocity(b_ang_velo);
        return_body->setFriction(o_friction);
        return_body->setRestitution(o_restitution);
        break;
    case 9:
        n_OtherBody->setMassProps(b_mass, b_inertia / incr_inertia_coef);
        return_body = n_OtherBody;
        break;
    case 10:
        n_OtherBody->setMassProps(b_mass, b_inertia / decr_inertia_coef);
        return_body = n_OtherBody;
        break;
    default:
        cout << "error¥n";
        break;
    }
    return return_body;
}
```

Code4 障害物の物理特性を変更する関数群

```
class MyContactResultCallback : public btCollisionWorld::ContactResultCallback
{
      ...
        const btRigidBody* specialBody = (body0->getUserIndex() == SPECIAL_INDEX) ? body0 : body1;
        const btRigidBody* otherBody = (body0->getUserIndex() == SPECIAL_INDEX) ? body1 : body0;
        time_t temp_t = clock() - touch_time;
        //0 はプレイヤー、1 は黒、99 は色源
```

```
        //一定時間が経過+触った先が変えれるなら...
        int user_index = otherBody->getUserIndex();
        if (user_index != 0 && user_index != 1 && (temp_t > 1000))
        {
            // Remove const modifier
            btRigidBody* n_OtherBody = const_cast<btRigidBody*>(otherBody);
            btRigidBody* n_SpecialBody = const_cast<btRigidBody*>(specialBody);
           ...
            // 筆の Index2 が 2(色吸収モード)のとき
            else if (n_SpecialBody->getUserIndex2() == 2)
            {
                n_OtherBody = RigidMakedWhite(n_SpecialBody, n_OtherBody);
                n_SpecialBody->setUserIndex2(n_OtherBody->getUserIndex());
                n_OtherBody->setUserIndex(2);
            }
            //筆の Index2 が塗りモードの場合(else if で競合回避)
            else if (3 <= n_SpecialBody->getUserIndex2() && n_SpecialBody->getUserIndex2() <= 10)
            {
                if (user_index == 2)
                {
                    n_OtherBody = SwitchColorChange(n_SpecialBody, n_OtherBody);
                    n_OtherBody->setUserIndex(n_SpecialBody->getUserIndex2());
                    n_SpecialBody->setUserIndex2(2);
                }
                //一回白に戻す必要がある(性質の重ね塗り対策)
                else
                {
                    n_OtherBody = RigidMakedWhite(n_SpecialBody, n_OtherBody);
                    n_OtherBody = SwitchColorChange(n_SpecialBody, n_OtherBody);
                    n_OtherBody->setUserIndex(n_SpecialBody->getUserIndex2());
                    n_SpecialBody->setUserIndex2(2);
                }
       ...
        return 0;
    }
};
```

Code5 障害物の物理特性を変更するプログラム

## 2.5 ステージの作成

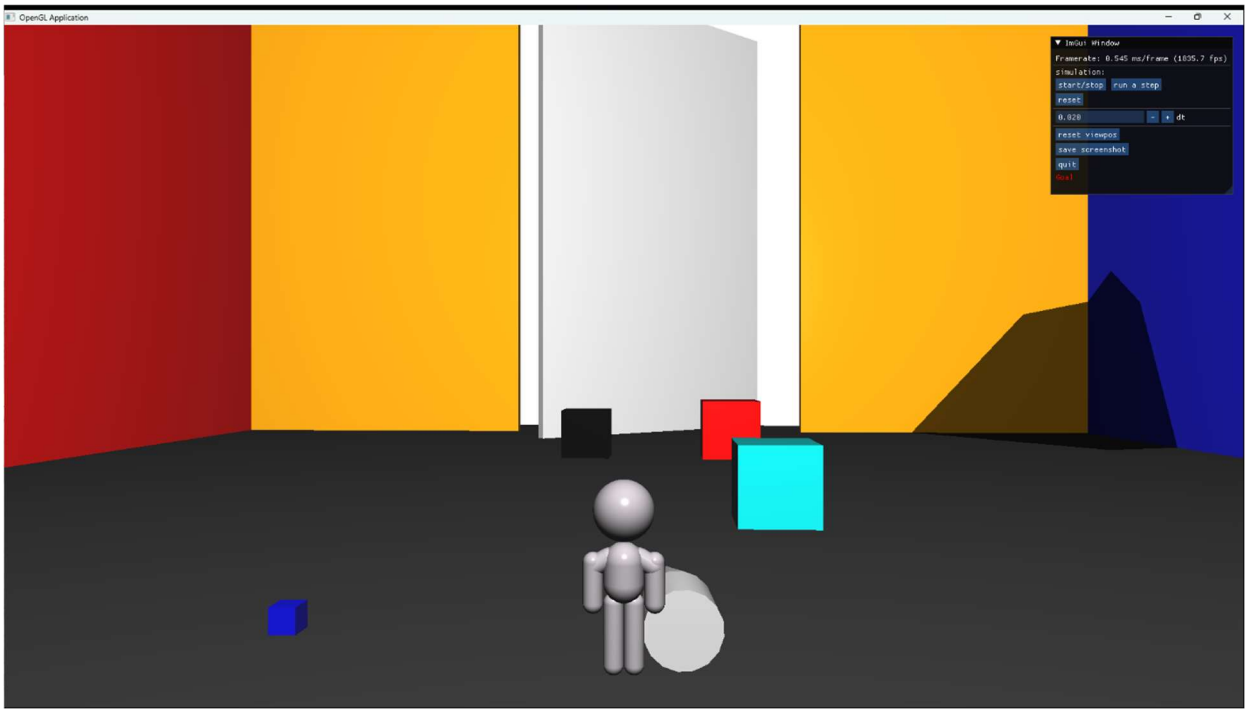


図 2-10 ステージ 1 の概観

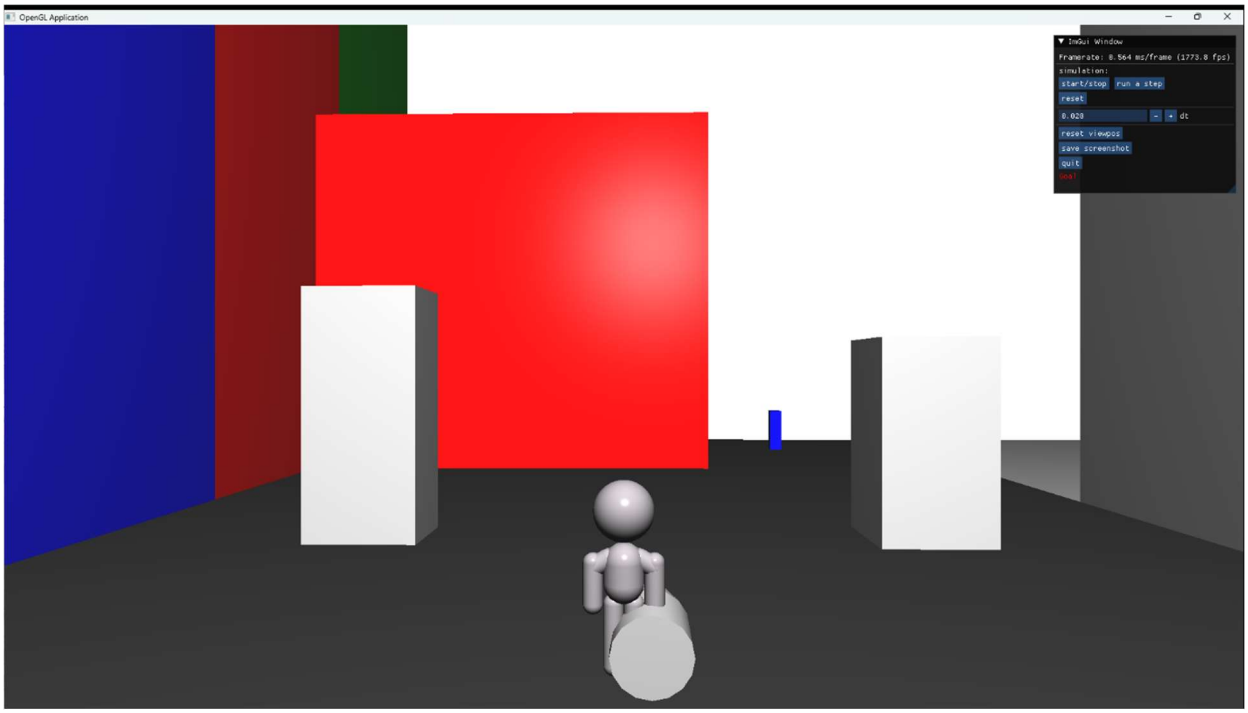


図 2-11 ステージ 2 の概観 1 枚目

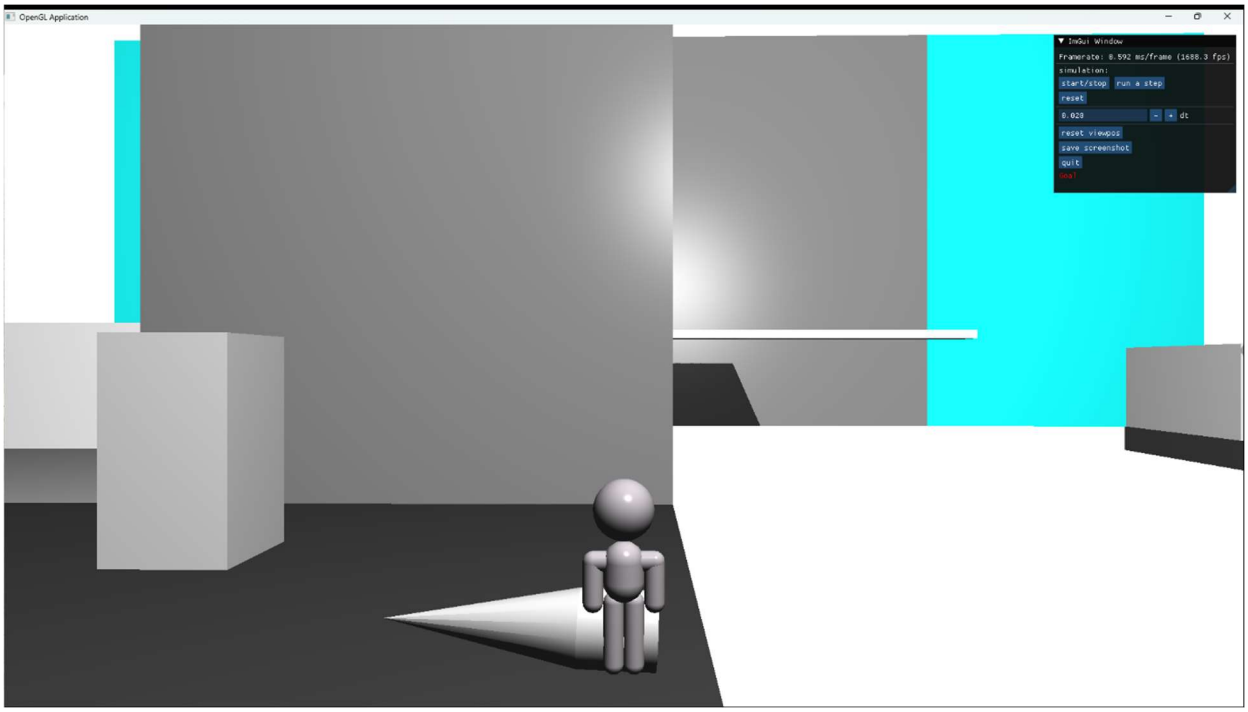


図 2-12 ステージ 2 の概観 2 枚目

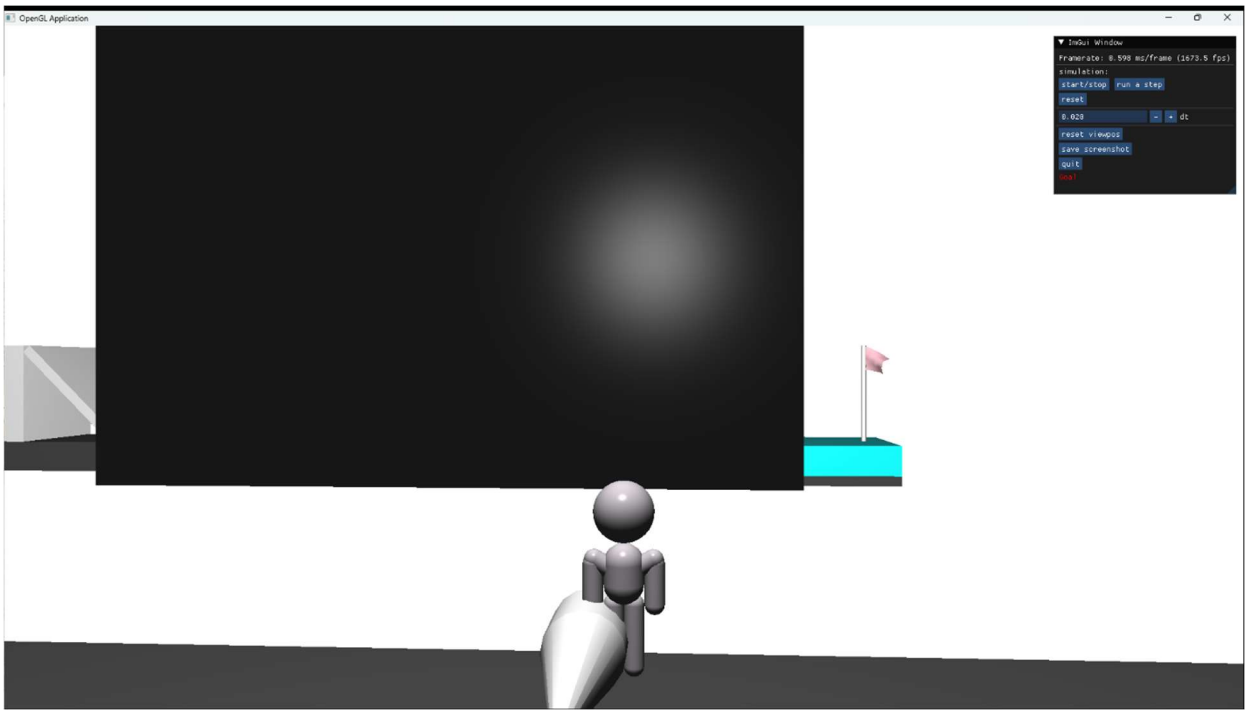


図 2-13 ステージ 2 の概観 3 枚目

### 2.5.1　説明

このゲームにはデモンストレーションとチュートリアルのために 2 つのステージが用意されている。ステージ 1 は、(図 2-10)に示すようなものである。これは、体験者に色と対応する物理特性の変化を確認してもらう狙いと、壁や床、筆のような特殊なオブジェクトの仕様を確認してもらう狙いがある。ステージ 2 は、(図 2-11)から(図 2-13)に示すような、色を変化させて解く簡易的な仕掛けの例を集めたものである。これは、本ゲームの面白さと問題点を明らかにする狙いがある。

### 2.5.2　機能の実装

以下の(Code6)によって実装した。

このコードは主にステージの構造とギミックを実装するために使用されており、2 つのステージを具体的に作成している。それぞれのステージ作成には、関数 MakeStage1 と MakeStage2 が使用されている。まず、関数 MakeStage1 では、ステージの基本パラメータを設定し、グラウンド、壁、障害物、キャラクター、旗などの要素を追加している。特に注目すべき点は、壁や障害物の配置に配列を使用しているため、コードを簡潔に記述していることである。次に、関数 MakeStage2 では、より複雑なステージを構築している。異なる高さのグラウンドや特殊な障害物、動く板などを追加することで、ステージの挑戦性と興味深さを高めている。特に、異なる方向に配置された壁や角度を持つ箱型障害物の追加などは、ステージ作成の柔軟性を示している。

こだわりのポイントは 2 つある。1 つ目は、各ステージごとに異なる構造と特色を持たせるために、関数内で詳細に各要素を設定している点である。2 つ目は、障害物の作成を形状または役割ごとに多数のパラメータを持つ関数としてまとめているため、今後のステージ作成を容易にしている点である。ざまざまな形状、役割の障害物の作成を行う関数は make_gimmick.cpp にまとめられている。

```
#include <random>
#include <iostream>
#include "make_stage.hpp"
#include "make_gimmick.hpp"
#include "utils.h"

using namespace std;

#define M_PI 3.1415926535

extern btRigidBody* rotate_rigid;
extern btSoftRigidDynamicsWorld* g_dynamicsworld;
extern btAlignedObjectArray<btCollisionShape*> g_collisionshapes;
extern btRigidBody* cp_rigid;
btVector3 p_respawn;
btVector3 b_respawn;
btVector3 min_viable_area;
btVector3 max_viable_area;
btScalar max_stage_x = 20;
btScalar max_stage_z = 20;
btScalar delta = 1.2;
btScalar half_wall_height = 5;
btScalar half_wall_width = 10;
```

```
void MakeStage1(void)
{
    min_viable_area = btVector3(-(max_stage_x + delta), -delta,-delta-2);
    max_viable_area = btVector3(delta, half_wall_height * 2, (max_stage_z + delta));
    //ステージごとにp_respawnを変更する
    AddGround(btVector3(-max_stage_x/2,- 0.3, max_stage_z/2-1), btVector3(max_stage_x/2, 0.3,
max_stage_z/2+1));
    btVector3 translations[5] = { btVector3(-max_stage_x,half_wall_height,max_stage_z/2),
btVector3(0,half_wall_height,max_stage_z/2), btVector3(-
max_stage_x/2,half_wall_height,max_stage_z),btVector3(-max_stage_x / 6, half_wall_height, 0),
btVector3(-max_stage_x / 6 * 5, half_wall_height, 0) };
    btVector3 sizes[5] = { btVector3(0.1,half_wall_height,half_wall_width),
btVector3(0.1,half_wall_height,half_wall_width) ,
btVector3(half_wall_width,half_wall_height,0.1),btVector3(max_stage_x / 6,half_wall_height,
0.1),btVector3(max_stage_x / 6,half_wall_height,0.1)};
    btQuaternion rotations[5];
    int colors[5] = { 7,8,97,10,10};
    for (int i = 0; i < 5; i++)
        AddWall(translations[i], sizes[i], colors[i]);
    btVector3 position = btVector3(-max_stage_x/2,half_wall_height, 0);
    btVector3 size = btVector3(max_stage_x / 6-0.1, half_wall_height, 0.1);
    btQuaternion rotation = btQuaternion(btVector3(1, 0, 0), 0);
    rotate_rigid=AddBoxBodies(200, position, size, rotation,2);
    AddHingeToSpace(rotate_rigid);
    RotateObstacle(rotate_rigid,10000);
    p_respawn = btVector3(-max_stage_x / 2, half_wall_height / 10, 4 * max_stage_z / 5);
    b_respawn = btVector3(-max_stage_x / 2, 1, 6 * max_stage_z / 10);
    CreateCharacter(p_respawn);
    CreateBrush(b_respawn);
    AddCheckFlag(btVector3(-15,0.5,-1),1, btVector3(20.0, 0.0, -20.0));
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> dis(-19, 19);
    for (int i = 1; i <= 10; ++i) {
        double x = dis(gen);
        while (x >= -1) {
            x = dis(gen);
        }
        double z = dis(gen);
        while (z <= 4) {
            z = dis(gen);
        }
        AddBoxBodies(1, btVector3(x, -0.2, z), btVector3(0.4, 0.4, 0.4), btQuaternion(0, 0, 0, 1), i);
    }
}

void MakeStage2(void)
{
    max_stage_z = 14;
    delta = 2.4;
    min_viable_area = btVector3(-(max_stage_x + delta), -delta, -(max_stage_z+delta));
    max_viable_area = btVector3(max_stage_x + delta, half_wall_height * 2,max_stage_z + delta);
    AddGround(btVector3(-15, -0.3, -1), btVector3(5, 0.3, 1));
    AddGround(btVector3(-15, -0.3, -8), btVector3(5, 0.3,6));
    AddGround(btVector3(-7, -0.7, -10), btVector3(3.0, 0.1, 4.0));
    AddGround(btVector3(-1.5,-0.7, -10), btVector3(1.5, 0.1, 4.0));
```

```
    AddGround(btVector3(-1, -0.3, 13), btVector3(3, 0.3, 1));
    AddGround(btVector3(-11, -0.7, 13), btVector3(7.0, 0.1, 1.0));
    AddGround(btVector3(-19, -0.7, 13), btVector3(1.0, 0.1, 1.0));
    AddWall(btVector3(-20.3, half_wall_height, -1), btVector3(0.3, half_wall_height, 1),4);
    AddWall(btVector3(-20.3, half_wall_height, -4), btVector3(0.3, half_wall_height, 2), 8);
    AddWall(btVector3(-20.3, half_wall_height, -8), btVector3(0.3, half_wall_height, 2), 7);
    AddWall(btVector3(-20.3, half_wall_height, -12), btVector3(0.3, half_wall_height, 2), 5);
    AddWall(btVector3(-9.7, half_wall_height, -3), btVector3(0.3, half_wall_height, 3), 9);
    AddWall(btVector3(2.3, half_wall_height, -7), btVector3(0.3, half_wall_height, 7), 3);
    AddWall(btVector3(2.3, half_wall_height,3.5), btVector3(0.3, half_wall_height, 3.5), 9);
    AddWall(btVector3(2.3, half_wall_height,10.5), btVector3(0.3, half_wall_height, 3.5), 3);
    //ジャンプの高さはだいたい0.8
    AddBoxBodies(100, btVector3(-12.5, 0.9, -4), btVector3(0.5,0.9,0.5));
    AddBoxBodies(100, btVector3(-17.5, 2.25, -12), btVector3(2.5,2.25,2.5),btQuaternion(0,0,0,1),7);
    AddBoxBodies(100, btVector3(-17.5, 1.125, -4), btVector3(0.5,1.125, 0.4));
    AddBoxBodies(100, btVector3(-12.5,0.36, -12), btVector3(0.5,1.8, 0.5), btQuaternion(0, 0, 0,
1),8);
    AddBoxBodies(1000, btVector3(-7,-0.3, -10), btVector3(3.0,0.3,4.0));
    AddBoxBodies(1000, btVector3(-1.5,0.9, -10), btVector3(1.5,1.5, 4.0));
    AddBoxBodies(1000, btVector3(0.0,1.0,13), btVector3(2.0,1.0,1.0));
    AddBoxBodies(0, btVector3(-3.0, 1.0, 13), btVector3(sqrt(2.0)-0.1,0.1,
1.0),btQuaternion(btVector3(0,0,1),btRadians(45.0)));
    AddBoxBodies(1000, btVector3(-11, -0.3, 13), btVector3(7.0, 0.3, 1.0));
    AddBoxBodies(1000, btVector3(-19, -0.3, 13), btVector3(1.0, 0.3, 1.0), btQuaternion(0, 0, 0,
1),3);
    AddBoxBodies(0, btVector3(-11,1.2, 13), btVector3(1.0, 0.3, 1.0), btQuaternion(0, 0, 0, 1), 2);
    AddBoxBodies(0, btVector3(-15,0.9, 13), btVector3(1.0, 0.1, 1.0), btQuaternion(0, 0, 0, 1), 2);
    p_respawn = btVector3(-15, 0.5, -1);
    b_respawn = btVector3(-15, 1.5, -1);
    AddMovingSkyBox(10, btVector3(1.0, 2.3, -10), btVector3(4, 0.1, 0.5), btVector3(0, 0, -14),
btVector3(0, 0, 10));
    AddBoxBodies(0, btVector3(-11,half_wall_height-0.8, 14.3), btVector3(7.0, half_wall_height, 0.3),
btQuaternion(0,0,0,1),1);
    AddBoxBodies(0, btVector3(-11,half_wall_height-0.8, 11.7), btVector3(7.0, half_wall_height, 0.3),
btQuaternion(0, 0, 0, 1), 1);
    CreateCharacter(p_respawn);
    CreateBrush(b_respawn);
    AddCheckFlag(btVector3(-19.5, 1.0, 13),2,btVector3(-20.0,0.0,-20.0));
}
```

Code6 ステージ作成を行う関数

# 3. 考察

## 3.1 本アプリケーションの総評

開発過程において、ゲーム内で使用される各オブジェクトの役割と実装が重要なテーマであった。具体的には、色の源として使用される壁、色を落とす床、色をつけることが可能な筆、色が変化する障害物などが挙げられる。これらの要素がバグ無しで実装できた点は評価できる。さらに、物理シミュレーションの特性を駆使しながら、現実とは異なる仮想世界の構築を行うという点も成功したと言えるだろう。しかし、このプロジェクトの全体評価としては、単なる成功とは言えない。後述す

るように改善点や問題点が多く存在すること、そして実装できなかった機能も複数あったことから、最終的な成果に対しては決して満足できるものではないと感じている。

## 3.2 改善点と問題点

### 3.2.1　色に対応した物理特性を覚えるのが難しい点

色に対応した物理特性が直感的でなく、初回プレイ時に特性を毎回確認する必要があるのは大きな問題だと言える。この点は、ユーザーエクスペリエンスを大きく損なう要素である。

この問題点の背後には、色と紐付けられたイメージが人によって異なる事実が存在している。例えば、黄色なら軽いという物理特性は、私の黄色に対するイメージから設定したものであるが、これは全ての人間の共通認識ではおそらく無いと思われる。この認識のギャップが初回プレイヤーの混乱を生み出していると考えられる。

改善点として提案される方向性は、色ではなくテクスチャが物理特性に影響を与える世界の構築である。テクスチャの利用は、色に比べればそのイメージの共通認識が存在するため、直感的に対応する物理特性を解釈しやすいという利点がある。この変更によって、プレイヤーによる認識のズレを最小限に抑え、ゲーム体験の質を向上させることが期待される。

### 3.2.2　別解が豊富にありすぎる点

物理特性を変化させる色が 8 色（障害物に使用できる 10 色のうち、未実装のピンク、塗り替えられない黒を除いた）存在するため、考えられる解法としては、ステージに存在する障害物の数を x 個とすると 8^x 通りの解法が生じることになる。この多岐にわたる解法を全て検証するのは、時間の観点から不可能であるため、一般公開する場合には大きな問題となると言える。

この問題の対策として効果的であると考えられるのは、そのステージに存在する色を限定することである。例えば、ステージ全体を通して使用される色が白ともう 1 色しかない場合には、解法は 2^x 通りとなり、検証が可能な範囲に収まる。このように、色の選択肢を制限することで、ゲームデザインの複雑さをコントロールし、プレイヤーに与える課題の明確化とプレイ体験の向上を図ることができるのではないかと考えられる。

### 3.2.3　パズルを考える/解くのに物理法則を理解している必要がある点

例えば、慣性モーメントだけを大きくした物体の挙動を想像するためには、剛体の回転運動の方程式を理解する必要がある。しかし、この方程式を理解している人は、社会人の中でも多くないと考えられる。この事実は、楽しめるユーザー層を大きく限定する意味で問題である。私自身も、慣性モーメントを大きくした際に瞬間回転速度が変化すると誤解していたため、実装したギミックが想定通りに動かず開発中に困難を経験した。このような問題を解消する改善点として、数学的な理解を要求する「質量の変化」「慣性モーメントの変化」「剛体から弾性体への変化」といった物理特性の変化は避ける方針が考えられる。

### 3.3 実装できなかった機能について

本プロジェクトにおける実装できなかった機能についての考察を展開すると、以下の 4 点が挙げられる。

まず 1 点目として、物体の色がピンクのとき弾性体に変化する機能である。この試みは、ゲームプレイに新たなダイナミクスを加えることを意図していたが、私の物理知識の不足により仕掛けを考えられなかったため組み入れなかった。

2 点目として、動作に合わせて部位を動かすことで実現するプレイヤーの簡易アニメーションである。この機能はプレイヤー体験の向上を目指していたが、ローカル座標と Constraint に対する知識不足によって実装が不可能であった。

3 点目として、複数の仕掛けを組み合わせて 1 つの巨大なパズルとするステージ 3 の作成も挑戦されたが、プロジェクト期間の制約により、この部分の実装は完遂されなかった。

4 点目として、マウスを右クリックしたときにプレイヤーの手を正面に突き出し、その際接触した物体を自由に動かす機能である。この機能はゲームの戦略性を高める要素であったが、こちらもローカル座標と Constraint に対する知識不足によって実装が不可能であった。

## 4. 実験の感想

まず、物理シミュレーションや衝突判定、constraint など、bullet の機能を多用したことが自分としては大変満足のいく結果であった。

次に、アプリケーションのコンセプト自体にスケールアップを阻害する要素が多く存在することが明らかになって良かったと感じる。

最後に、プレイヤーのパーツを動かすことで簡易的なアニメーションを実装したり、プレイヤーの手と接触した物体を自在に動かす技術が導入されていれば、より見栄えの良い、自由度の高いパズルの作成が可能であったと考えられる。しかし、残念ながら、そこまでの実装は手が回らなかった。今回の経験から得た教訓は、今後のソフトウェア開発においても重要な指針となるだろう。

## 5. 参考文献


1) 藤沢誠.n.d.物理エンジンを使ったアプリケーション開発.GitHub.n.d. https://fujis.github.io/iml_physics/.2023/08/08
2) Bulletphysics.org.n.d. Bullet Collision Detection & Physics Library.Bullet.n.d. https://pybullet.org/Bullet/BulletFull/.2023/08/08