1　目的

Game プログラムの開発を通して、オブジェクト指向プログラミングに関する技術を習得する。

2　システムの概要

私が作成した新キャラクターは隠しブロックである．このキャラクターは最初は現れておらず，隠しブロックがある場所に Player が上向きの慣性をもった状態で衝突することで出現する．出現時，Player は隠しブロックより上に行くことはせず，速度を失い落ちていく．それと同時に Player はスコアを 20 手に入れる．出現後は Platform と同じように Player が下向きの慣性を持っている状態で newCharacter に衝突することで Player がジャンプする．また，出現時は図 1 のような姿をしている．

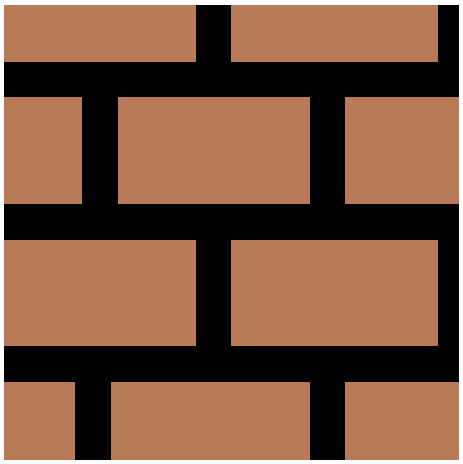

図 1: 新キャラクター出現時イメージ

3　システムの構成

新キャラクターの関係図を図 2 に示す.

図 2: 新キャラクタークラス図

新キャラクターを定義するクラス **NewCharacter** は，**GameCharacter** を継承し，**Player**，**World0** と接続している．

属性

新キャラクターは **GameCharacter** から継承したものも含め，ソースコード 1 に示すようなカプセル化が行われた属性を持っている．

ソースコード 1: 属性

```
1  int x;
2  int y;
3  int xSize;
4  int ySize;
5  boolean disappearFlag;
6  Player player;
```

x,y,xSize,ySize は **NewCharacter** のオブジェクトの位置と大きさの定義である．

disappearFlag は **NewCharacter** のオブジェクトがプレイヤーの画面から隠れているか否かを管理する変数で，true の時隠れていて false で出現している．初期値は true である．

player は **Player** と接続し利用するための変数である．

メソッド

GameCharacter のメソッドをソースコード 2 に示す．

ソースコード 2: メソッド

```
1  public void setPlayer(Player player) {
2    this.player = player;
3  }
4
5  public boolean getDisappearFlag(){
6    return disappearFlag;
7  }
8
9  public void move(){
10   // 隠しボックスが取得された後は足場として利用
11   if(disappearFlag==false && overlap(player) == true &&
         player.getySpeed()<0 && player.getHeadingFlag()==false)
         {
12     player.jump();
13   }
14   // 隠しボックスの取得
15   if(getDisappearFlag()==true && overlap(player) == true &&
         player.getySpeed()>0){//下方向から重なった場合
         のみ
16     player.heading();
17     player.addPoint(20);
18     disappearFlag = false;
19   }
20 }
```

setPlayer(),getDisappearFlag() は move() で使うために定義したセッターとゲッターである．move() の処理は **NewCharacter** のオブジェクトが **Player** のオブジェクトと重なった場合についての処理が記述してある．disappearFlag=true で隠れている場合，**Player** のオブジェクトがジャンプして上向きの速度を持っている状態で **NewCharacter** のオブジェクトと重なった場合にそのオブジェクトを取得するものである．取得時の処理として，**Player** オブジェクトが頭をぶつける処理 player.heading()，スコアが 20 加算される処理

player.addPoint(20)，**NewCharacter** のオブジェクトが叩かれたことにする処理を行っている．

heading() は隠れている **NewCharacter** のオブジェクトを叩いた **Player** のオブジェクトが接触し，**NewCharacter** のオブジェクトが出現した後すぐにジャンプしてしまったため **Player** に作成したメソッドである．このメソッドをソースコード 3 に示す.

ソースコード 3: heading()

```
1 private boolean headingFlag = false;
2 public void heading(){
3   headingFlag = true;
4   ySpeed = 0.0f;
5 }
```

このメソッドは呼び出された時に headingFlag を上げ，**Player** のオブジェクトの y 軸方向のスピードを 0 にする処理出できている．headingFlag は **Player** のオブジェクトがジャンプしたときに下ろされる様になっており，前述の問題を解決している．なお，ばねによるハイジャンプでは解除されない仕様になっている．

次に，disappearFlag=false で出現している場合，**Player** のオブジェクトが **NewCharacter** のオブジェクトに触れて出現した後，一回以上ジャンプした上で **Player** のオブジェクト自身が下向きの速度を持っている状態で，**NewCharacter** のオブジェクトと重なった場合にそのオブジェクトでジャンプするものである．

4 考察

4.1 オブジェクト指向プログラミングの長所

オブジェクト指向プログラミングでは継承と多態性により，似たようなオブジェクトを複数作る際にクラス内のメンバやメソッドを共通化し，少ない記述量で作成できる．また，各クラスに役割を分割することで，エラーが起きた際にエラー内容から原因のファイルが推測しやすくなる．

4.2 オブジェクト指向プログラミングで注意すべき点

オブジェクト指向プログラミングでは設計が重要だが，設計を過剰に複雑化しすぎると，チームの仲間や未来の自分がコードの理解や保守が難しくなり時間がかかってしまう．オブジェクト指向プログラミングの利点をうまく利用できれば保守性や拡張性に優れたソフトウェアを開発できる反面，上手

く扱えなければそれがバグの温床になったり可読性の低下などマイナスに働きかねない.

5　感想

聞き逃してしまったときに解説しているスライドが配布されていない時がたまにあったのが不便だった．今回初めて 1 からアプリ開発をしたが，順序立てて少しずついろいろな機能を追加していったので，オブジェクト指向についてよく学べたと思う.