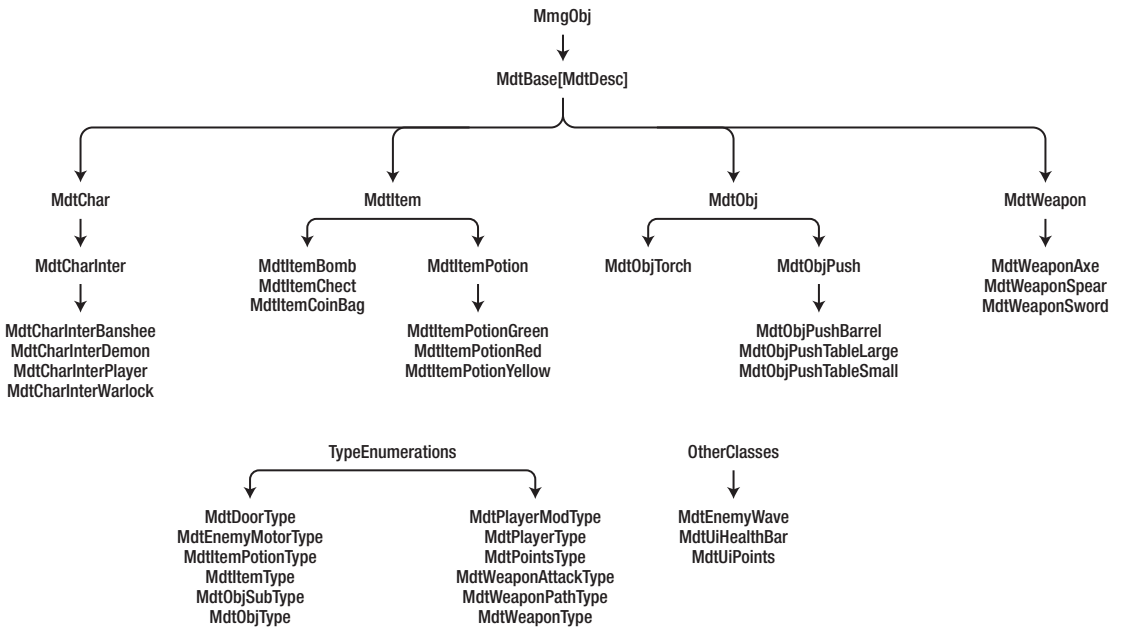
CHAPTER 23

DungeonTrap Level 1 and 2 Extended Classes

In this chapter, we're going to implement the higher-level classes that extend the base classes we've just finished reviewing in Chapter 22. I wanted to show the class inheritance diagram again.



this figure will be printed in b/w

Figure 23-1. *MdtCharInter Class Inheritance Diagram 1*

The classes we'll review as part of the level 1 extended classes are the `MdtCharInter`, `MdtItemPotion`, `MdtItemCoinBag`, `MdtObjPush`, `MdtObjTorch`, and `MdtWeaponSpear` classes. The level 2 extended classes that we'll review here are the `MdtCharInterDemon`,

V. G. Brusca, *Introduction to Video Game Engine Development*, https://doi.org/10.1007/978-1-4842-7039-4_23

`MdtItemPotionGreen`, `MdtObjPushBarrel`, and `MdtCharInterPlayer` classes. The new functionality provided by these classes should cover the game specifications that mention using items, objects, and weapons to fight off waves of enemies.

DungeonTrap: Level 1 Classes

We'll cover the level 1 extended classes first. I've listed them in the following group. Some of the classes are in fact new base classes that are extended by level 2 classes in the diagram:

- MdtCharInter (new base class)
- MdtItemPotion (new base class)
- MdtItemCoinBag
- MdtObjPush (new base class)
- MdtWeaponSpear

We'll start the class reviews with the `MdtCharInter` class, which stands for "character interactive." This extension of the `MdtChar` class supports breaking and bouncing the characters. Breaking refers to the animation played when an enemy takes lethal damage. Note that this class becomes a new base for player and enemy character classes.

MdtCharInter: Class Review

The `MdtCharInter` class, as noted in the preceding, stands for "character interactive"; and it's used to define a game player that can be bounced, take lethal damage, and be broken. This class builds on the functionality provided by the `MdtChar` class. The `MdtCharInter` class has no pertinent static class members or enumerations to speak of, so we'll begin the class review with the class' fields.

MdtCharInter: Class Fields

The `MdtCharInter` class builds on the functionality of the `MdtChar` class by adding fields to help track new features like breaking and bouncing a character. Let's take a look.

Listing 23-1. MdtCharInter Class Fields 1

```
public MdtPlayerType playerType = MdtPlayerType.NONE;
public MmgSprite subjBreaks = null;
public boolean isBroken = false;
public MdtPlayerType brokenBy;
```

The first block of class fields are listed in the preceding. The `playerType` field is used to mark what kind of character the player is. The next entry, the `subjBreaks` class field, is used to display the character's death animation, in this case referred to as "breaking." A Boolean flag, `isBroken`, is used to indicate that the character has been defeated; and the breaking animation should begin. The last entry listed in the preceding is the `brokenBy` field. This field records which player type broke the current character.

Listing 23-2. MdtCharInter Class Fields 2

```
public boolean isBouncing = false;
public int bounceDirOrig = 0;
public int bounceDirX = 0;
public int bounceDirY = 0;
public long bouncingCurrentMs;
public long bouncingLengthMs = 175;
```

The first field in the preceding list, `isBouncing`, is a Boolean flag that indicates if the current character is bouncing. A character is bounced when it's hit by an enemy, a thrown object, or a player's weapon. The next field is used to track the original direction of the object that caused the bounce. The next two fields are used to track the direction, on the X and Y axes, that the character is bounced in.

The `bouncingCurrentMs` field is used to track the total time elapsed during the current bounce animation. The last field listed, `bouncingLengthMs`, represents the total time the character can spend bouncing.

Listing 23-3. MdtCharInter Class Fields 3

```
public MdtEnemyMotorType motor = MdtEnemyMotorType.NONE;
private long motorMoveMs = 0;
private long motorMoveLengthMs = 350;
private MdtPlayerType targetPlayer = MdtPlayerType.NONE;
```

The last block of class fields that we have to review are listed in the preceding. The `motor` field is used to define which simple 2D AI is used to drive the character's movement. This feature is only enabled for enemy characters. The `motorMoveMs` field tracks how much time that the current character movement has been running.

The `motorMoveLengthMs` field indicates how much time can be spent moving this iteration. The timing values are used to toggle movement in a given direction for an interval of time. That wraps up our review of the `MdtCharInter` class fields. Lastly, enemy characters need to have a target to fight, and the `targetPlayer` indicates which player the current enemy character is attacking. Up next, we'll take a look at the class' method outline.

MdtCharInter: Pertinent Method Outline

The `MdtCharInter` class method outline is listed in the following with the methods separated into two groups, main and support methods.

Listing 23-4. MdtCharInter Pertinent Method Outline 1

```
//Main Methods
public MdtCharInter(MmgSprite Subj, int FrameFrontS, int FrameFrontE,
int FrameBackS, int FrameBackE, int FrameLeftS, int FrameLeftE, int
FrameRightS, int FrameRightE, ScreenGame Screen, MdtObjType ObjType,
MdtObjSubType ObjSubType) { ... }

public void Bounce(MmgVector2 collPos, int halfWidth, int halfHeight, int
bounceDir, MdtPlayerType BounceBy) { ... }


public void SetDir(int d) { ... }
public void SetDirSafe(int d) { ... }
public void TakeDamage(int i, MdtPlayerType p) { ... }

public void SetPosition(MmgVector2 v) { ... }
public void SetPosition(int x, int y) { ... }
public void SetX(int i) { ... }
public void SetY(int i) { ... }
public boolean MmgUpdate(int updateTick, long currentTimeMs, long
msSinceLastFrame) { ... }
```

```
public boolean MmgUpdate(int updateTick, long currentTimeMs, long
msSinceLastFrame) { ... }

//Support Methods
public MdtPlayerType GetPlayerType() { ... }
public void SetPlayerType(MdtPlayerType p) { ... }
public MmgSprite GetSubjBreaks() { ... }
public void SetSubjBreaks(MmgSprite s) { ... }
public boolean GetIsBroken() { ... }
public void SetIsBroken(boolean b) { ... }
public MdtPlayerType GetBrokenBy() { ... }
public void SetBrokenBy(MdtPlayerType p) { ... }
public boolean GetIsBouncing() { ... }
public void SetIsBouncing(boolean b) { ... }
public long GetBouncingCurrentMs() { ... }
public void SetBouncingCurrentMs(long l) { ... }
public long GetBouncingLengthMs() { ... }
public void SetBouncingLengthMs(long l) { ... }
public MdtEnemyMotorType GetMotor() { ... }
public void SetMotor(MdtEnemyMotorType m) { ... }
```

We'll be reviewing the methods listed in the preceding in some detail, but before we get into the method review, I wanted to list the definitions for both the Java and C# versions of the class.

Listing 23-5. MdtCharInter Class Definitions 1

```
//Java Version
public class MdtCharInter extends MdtChar {

//C# Version
public class MdtCharInter : MdtChar {
```

You can add a new class to your project named `MdtCharInter`, alongside the project's other class files. Carefully add in the class fields reviewed here or copy and paste them in from the completed version of the class in the completed chapter project contained in the game engine's project folder. Let's take a look at the class' support methods next.

MdtCharInter: Support Method Details

The `MdtCharInter` class' support methods are basic get and set methods. I'll list them here, but we won't go into any detail reviewing them due to their simplicity.

Listing 23-6. MdtCharInter Support Method Details 1

```
01 public MdtPlayerType GetPlayerType() {
02     return playerType;
03 }

01 public void SetPlayerType(MdtPlayerType p) {
02     playerType = p;
03 }

01 public MmgSprite GetSubjBreaks() {
02     return subjBreaks;
03 }

01 public void SetSubjBreaks(MmgSprite s) {
02     subjBreaks = s;
03 }

01 public boolean GetIsBroken() {
02     return isBroken;
03 }

01 public void SetIsBroken(boolean b) {
02     isBroken = b;
03 }

01 public MdtPlayerType GetBrokenBy() {
02     return brokenBy;
03 }

01 public void SetBrokenBy(MdtPlayerType p) {
02     brokenBy = p;
03 }

01 public boolean GetIsBouncing() {
```

```
02      return isBouncing;
03  }

01  public void SetIsBouncing(boolean b) {
02      isBouncing = b;
03  }

01  public long GetBouncingCurrentMs() {
02      return bouncingCurrentMs;
03  }

01  public void SetBouncingCurrentMs(long l) {
02      bouncingCurrentMs = l;
03  }

01  public long GetBouncingLengthMs() {
02      return bouncingLengthMs;
03  }

01  public void SetBouncingLengthMs(long l) {
02      bouncingLengthMs = l;
03  }

01  public MdtEnemyMotorType GetMotor() {
02      return motor;
03  }

01  public void SetMotor(MdtEnemyMotorType m) {
02      motor = m;
03  }
```

Take the time to review and understand the support methods listed here before adding them to your class file. I'll provide instructions on how to copy and paste all level 1 class files into your project, to save you some time, at the end of the level 1 class reviews. In the next section, we'll tackle the class' main methods.

MdtCharInter: Main Method Details

In this section, we'll take a detailed look at the class' main methods. The first block of methods for us to review is listed in the following group.

Listing 23-7. MdtCharInter Main Method Details 1

```
01 public MdtCharInter(MmgSprite Subj, int FrameFrontS, int FrameFrontE,
   int FrameBackS, int FrameBackE, int FrameLeftS, int FrameLeftE, int
   FrameRightS, int FrameRightE, ScreenGame Screen, MdtObjType ObjType,
   MdtObjSubType ObjSubType) {
02     super(Subj, FrameFrontS, FrameFrontE, FrameLeftS, FrameLeftE,
       FrameRightS, FrameRightE, FrameBackS, FrameBackE, Screen, ObjType,
       ObjSubType);
03
04     MmgBmp src = MmgHelper.GetBasicCachedBmp("explosion_anim_
       spritesheet_lg.png");
05     MmgSpriteSheet ssSrc = new MmgSpriteSheet(src, 32, 32);
06     subjBreaks = new MmgSprite(ssSrc.GetFrames());
07     subjBreaks.SetMsPerFrame(50);
08
09     weapons = new Hashtable();
10     weapons.put("sword", new MdtWeaponSword(this, MdtWeaponType.SWORD,
       MdtPlayerType.ENEMY));
11     weapons.put("axe", new MdtWeaponAxe(this, MdtWeaponType.AXE,
       MdtPlayerType.ENEMY));
12     weapons.put("spear", new MdtWeaponSpear(this, MdtWeaponType.SPEAR,
       MdtPlayerType.ENEMY));
13
14     weaponCurrent = weapons.get("spear");
15     weaponCurrent.SetHolder(this);
16     weaponCurrent.active = true;
17 }
```

```
01 public void Bounce(MmgVector2 collPos, int halfWidth, int halfHeight,
   int bounceDir, MdtPlayerType BounceBy) {
02     bounceDirOrig = bounceDir;
```

```
03      isMoving = false;
04
05      bouncingCurrentMs = 0;
06      isBouncing = true;
07
08      if(bounceDir == MmgDir.DIR_LEFT || bounceDir == MmgDir.DIR_RIGHT) {
09          if(collPos.GetY() + halfHeight >= GetY() + GetHeight()/2) {
10              bounceDirX = bounceDir;
11              bounceDirY = MmgDir.DIR_BACK;
12          } else {
13              bounceDirX = bounceDir;
14              bounceDirY = MmgDir.DIR_FRONT;
15          }
16      } else if(bounceDir == MmgDir.DIR_FRONT || bounceDir == MmgDir.
        DIR_BACK) {
17          if(collPos.GetX() + halfWidth >= GetX() + GetWidth()/2) {
18              bounceDirX = MmgDir.DIR_LEFT;
19              bounceDirY = bounceDir;
20          } else {
21              bounceDirX = MmgDir.DIR_RIGHT;
22              bounceDirY = bounceDir;
23          }
24      }
25
26      if(playerType == MdtPlayerType.ENEMY) {
27          TakeDamage(1, BounceBy);
28      }
29  }
```

```
@Override
01  public void SetDir(int d) {
02      if(d == MmgDir.DIR_FRONT) {
03          subj.SetFrameStart(frameFrontStart);
04          subj.SetFrameStop(frameFrontStop);
05          subj.SetFrameIdx(frameFrontStart);
06      } else if(d == MmgDir.DIR_BACK) {
```

```
07          subj.SetFrameStart(frameBackStart);
08          subj.SetFrameStop(frameBackStop);
09          subj.SetFrameIdx(frameBackStart);
10      } else if(d == MmgDir.DIR_LEFT) {
11          subj.SetFrameStart(frameLeftStart);
12          subj.SetFrameStop(frameLeftStop);
13          subj.SetFrameIdx(frameLeftStart);
14      } else if(d == MmgDir.DIR_RIGHT) {
15          subj.SetFrameStart(frameRightStart);
16          subj.SetFrameStop(frameRightStop);
17          subj.SetFrameIdx(frameRightStart);
18      }
19      dir = d;
20  }
```

```
01  public void SetDirSafe(int d) {
02      if(GetDir() != d) {
03          SetDir(d);
04      }
05  }
```

```
@Override
01  public void TakeDamage(int i, MdtPlayerType p) {
02      super.TakeDamage(i, p);
03      SetBrokenBy(p);
04  }
```

The first main method listed is the class constructor. Note that the C# version of the file uses the `base` class syntax, not the `super` class syntax found in the Java version of the class. Refer to the chapter's completed project code if you run into trouble. In any case, the constructor initializes the super class fields using the values provided by the constructor arguments.

Class-specific fields are initialized on subsequent lines of code. An `MmgSprite` animation is loaded on lines 4–7 and is used to indicate the character has received critical damage. On lines 9–16, the weapons are loaded and configured to the default weapon, the spear. Notice that the game has more functionality than is active in the game. There are a lot of features I've left for you to play around with once you get the base completed.

The second method for us to review is the Bounce method. This method is used to bounce the player in the opposite direction of a collision. The code on lines 2–6 is used to set the character's state. On lines 8–24, the actual bounce direction is calculated for the X and Y coordinates. Notice that the resulting bounce is determined by the point of collision with the object causing the bounce.

The third main method to look at is the SetDir method. This method is responsible for setting the direction of the character and is configured to set the start and stop frames for the character's walking animation in the given direction. The SetDirSafe method is similar to the SetDir method except that it only updates the direction if the current direction is different than the desired one. This allows the character to animate its frames as well as its direction while moving.

The last method listed in the preceding set is the TakeDamage method. This method is an override of the MdtChar implementation of the method. In this case, we're keeping track of which character type has damaged the current character. The main difference in this implementation is that the brokenBy field is updated each time damage is taken. The next block of main methods for us to look at are the overridden positioning methods. Let's take a look at some code.

Listing 23-8. MdtCharInter Main Method Details 2

```
@Override
01 public void SetPosition(MmgVector2 v) {
02    super.SetPosition(v);
03     subj.SetPosition(v);
04     subjBreaks.SetPosition(v.GetX() + (subj.GetWidth() - subjBreaks.
       GetWidth())/2, v.GetY() + (subj.GetHeight() - subjBreaks.
       GetHeight())/2);
05 }

@Override
01 public void SetPosition(int x, int y) {
02     super.SetPosition(x, y);
03     subj.SetPosition(x, y);
04     subjBreaks.SetPosition(x + (subj.GetWidth() - subjBreaks.
       GetWidth())/2, y + (subj.GetHeight() - subjBreaks.GetHeight())/2);
05 }
```

```
@Override
01 public void SetX(int i) {
02     super.SetX(i);
03     subj.SetX(i);
04     subjBreaks.SetX(i + (subj.GetWidth() - subjBreaks.GetWidth())/2);
05 }

@Override
01 public void SetY(int i) {
02     super.SetY(i);
03     subj.SetY(i);
04     subjBreaks.SetY(i + (subj.GetHeight() - subjBreaks.GetHeight())/2);
05 }
```

As you've seen before, the overridden positioning methods are meant to keep the position of the object, the subject, and the subject breaking animation synchronized. Notice that the `subjBreaks`, an `MmgSprite` instance, is positioned at the center of the `subj` field.

The last block of main methods for us to review contains the game engine drawing routine methods, the `MmgUpdate` and `MmgDraw` methods. In this case, the `MmgUpdate` method is responsible for providing the bounce and break animations. Let's take a look at some code!

Listing 23-9. MdtCharInter Main Method Details 3

```
@Override
001 public boolean MmgUpdate(int updateTick, long currentTimeMs, long
    msSinceLastFrame) {
002     lret = false;
003     if (isVisible == true) {
004         if(isBroken) {
005             subjBreaks.MmgUpdate(updateTick, currentTimeMs,
                msSinceLastFrame);
006             if(subjBreaks.GetFrameIdx() == subjBreaks.GetFrameStop()) {
007                 if(GetPlayerType() == MdtPlayerType.ENEMY && GetRand().
                    nextInt(11) % 2 == 0) {
008                     screen.UpdateGenerateItem(GetX(), GetY());
009                 }
12
```

```
010                 screen.UpdateRemoveObj(this, brokenBy);
011             }
012         } else {
013             if(healthCurrent <= 0) {
014                 isBroken = true;
015             }
016 
017             subj.MmgUpdate(updateTick, currentTimeMs, msSinceLastFrame);
018 
019             if(!isAttacking) {
020                 if(isBouncing) {
021                     bouncingCurrentMs += msSinceLastFrame;
022                     if(bouncingCurrentMs <= bouncingLengthMs) {
023                         current = new MmgRect(subj.GetX(), subj.GetY(),
                            subj.GetY() + subj.GetHeight(), subj.GetX() +
                            subj.GetWidth());
024                         if(speed < 0) {
025                             speed *= -1;
026                         }
027 
028                         int nX = 0;
029                         int nY = 0;
030 
031                         if(bounceDirX == MmgDir.DIR_LEFT) {
032                             nX = (speed * -2);
033                         } else if(bounceDirX == MmgDir.DIR_RIGHT) {
034                             nX = (speed * 2);
035                         }
036 
037                         if(bounceDirY == MmgDir.DIR_BACK) {
038                             nY = (speed * -2);
039                         } else if(bounceDirY == MmgDir.DIR_FRONT) {
040                             nY = (speed * 2);
041                         }
042 
043                         if(bounceDirY == MmgDir.DIR_BACK) {
```

```
044                             if(subj.GetY() - (speed * 2) >= ScreenGame.
                                BOARD_TOP) {
045                                 current.ShiftRect(nX, nY);
046                                 coll = screen.CanMove(current, this);
047                                 if(coll == null) {
048                                     SetY(current.GetTop());
049                                 } else if(coll.GetMdtType() ==
                                    MdtObjType.PLAYER) {
050                                     ((MdtCharInter)coll).
                                        Bounce(GetPosition(), GetWidth()/2,
                                        GetHeight()/2, bounceDirOrig,
                                        playerType);
051                                 } else if(coll.GetMdtType() ==
                                    MdtObjType.ENEMY) {
052                                     ((MdtCharInter)coll).
                                        Bounce(GetPosition(), GetWidth()/2,
                                        GetHeight()/2, bounceDirOrig,
                                        playerType);
053                                 } else if(coll.GetMdtType() ==
                                    MdtObjType.OBJECT) {
054                                     //stop
055                                 } else {
056                                     SetY(current.GetTop());
057                                 }
058                             } else {
059                                 SetY(ScreenGame.BOARD_TOP);
060                             }
061                         } else if(bounceDirY == MmgDir.DIR_FRONT) {
062                             if(subj.GetY() + subj.GetHeight() +
                                (speed * 2) <= ScreenGame.BOARD_BOTTOM) {
063                                 current.ShiftRect(nX, nY);
064                                 coll = screen.CanMove(current, this);
065                                 if(coll == null) {
066                                     SetY(current.GetTop());
067                                 } else if(coll.GetMdtType() ==
                                    MdtObjType.PLAYER) {
```

```
068                         ((MdtCharInter)coll).
                            Bounce(GetPosition(), GetWidth()/2,
                            GetHeight()/2, bounceDirOrig,
                            playerType);
069                     } else if(coll.GetMdtType() ==
                        MdtObjType.ENEMY) {
070                         ((MdtCharInter)coll).
                            Bounce(GetPosition(), GetWidth()/2,
                            GetHeight()/2, bounceDirOrig,
                            playerType);
071                     } else if(coll.GetMdtType() ==
                        MdtObjType.OBJECT) {
072                         //stop
073                     } else {
074                         SetY(current.GetTop());
075                     }
076                 } else {
077                     SetY(ScreenGame.BOARD_BOTTOM - subj.
                        GetHeight());
078                 }
079             }
080
081             if(bounceDirX == MmgDir.DIR_LEFT) {
082                 if(subj.GetX() - (speed * 2) >= ScreenGame.
                    BOARD_LEFT) {
083                     current.ShiftRect(nX, nY);
084                     coll = screen.CanMove(current, this);
085                     if(coll == null) {
086                         SetX(current.GetLeft());
087                     } else if(coll.GetMdtType() ==
                        MdtObjType.PLAYER) {
088                         ((MdtCharInter)coll).
                            Bounce(GetPosition(), GetWidth()/2,
                            GetHeight()/2, bounceDirOrig,
                            playerType);
```

```
089                                 } else if(coll.GetMdtType() ==
                                    MdtObjType.ENEMY) {
090                                     ((MdtCharInter)coll).
                                        Bounce(GetPosition(), GetWidth()/2,
                                        GetHeight()/2, bounceDirOrig,
                                        playerType);
091                                 } else if(coll.GetMdtType() ==
                                    MdtObjType.OBJECT) {
092                                     //stop
093                                 } else {
094                                     SetX(current.GetLeft());
095                                 }
096                             } else {
097                                 SetX(ScreenGame.BOARD_LEFT);
098                             }
099                         } else if(bounceDirX == MmgDir.DIR_RIGHT) {
100                             if(subj.GetX() + subj.GetWidth() +
                                (speed * 2) <= ScreenGame.BOARD_RIGHT) {
101                                 current.ShiftRect(nX, nY);
102                                 coll = screen.CanMove(current, this);
103                                 if(coll == null) {
104                                     SetX(current.GetLeft());
105                                 } else if(coll.GetMdtType() ==
                                    MdtObjType.PLAYER) {
106                                     ((MdtCharInter)coll).
                                        Bounce(GetPosition(), GetWidth()/2,
                                        GetHeight()/2, bounceDirOrig,
                                        playerType);
107                                 } else if(coll.GetMdtType() ==
                                    MdtObjType.ENEMY) {
108                                     ((MdtCharInter)coll).
                                        Bounce(GetPosition(), GetWidth()/2,
                                        GetHeight()/2, bounceDirOrig,
                                        playerType);
109                                 } else if(coll.GetMdtType() ==
                                    MdtObjType.OBJECT) {
```

```
110                                 //stop
111                             } else {
112                                 SetX(current.GetLeft());
113                             }
114                         } else {
115                             SetX(ScreenGame.BOARD_RIGHT - subj.
                                GetWidth());
116                         }
117                     }
118                 } else {
119                     isBouncing = false;
120                     bouncingCurrentMs = 0;
121                 }
122             }
123
124             if(motor != MdtEnemyMotorType.NONE && playerType ==
                MdtPlayerType.ENEMY) {
125                 MmgVector2 mPos = null;
126                 if(targetPlayer == MdtPlayerType.NONE) {
127                     if(screen.GetGameType() == GameType.GAME_TWO_
                        PLAYER && !screen.GetPlayer2Broken()) {
128                         int t = GetRand().nextInt(11);
129                         if(t % 2 == 0) {
130                             targetPlayer = MdtPlayerType.PLAYER_1;
131                         } else {
132                             targetPlayer = MdtPlayerType.PLAYER_2;
133                         }
134                     } else {
135                         targetPlayer = MdtPlayerType.PLAYER_1;
136                     }
137                 }
138
139                 if(screen.GetGameType() == GameType.GAME_ONE_
                    PLAYER) {
140                     mPos = screen.GetPlayer1Pos();
```

```
141                     } else if(screen.GetGameType() == GameType.GAME_
                        TWO_PLAYER) {
142                         if(targetPlayer == MdtPlayerType.PLAYER_1 &&
                            !screen.GetPlayer1Broken()) {
143                             mPos = screen.GetPlayer1Pos();
144                         } else if(targetPlayer == MdtPlayerType.
                            PLAYER_2 && !screen.GetPlayer2Broken()) {
145                             mPos = screen.GetPlayer2Pos();
146                         }
147                     }
148
149                     if(mPos != null) {
150                         motorMoveMs += msSinceLastFrame;
151                         if(motorMoveMs >= motorMoveLengthMs) {
152                             int t = GetRand().nextInt(11);
153                             if(t % 3 == 0) {
154                                 isMoving = true;
155                             } else {
156                                 isMoving = false;
157                             }
158                             motorMoveMs = 0;
159                         }
160
161                         if(motor == MdtEnemyMotorType.MOVE_X_THEN_Y) {
162                             if(GetX() + GetWidth()/2 < mPos.GetX()) {
163                                 SetDirSafe(MmgDir.DIR_RIGHT);
164                             } else if(GetX() > mPos.GetX() +
                                GetWidth()/2) {
165                                 SetDirSafe(MmgDir.DIR_LEFT);
166                             } else if(GetY() + GetHeight()/2 < mPos.
                                GetY()) {
167                                 SetDirSafe(MmgDir.DIR_FRONT);
168                             } else {
169                                 SetDirSafe(MmgDir.DIR_BACK);
170                             }
171                       } else if(motor == MdtEnemyMotorType.MOVE_Y_THEN_X) {
18
```

```
172                         if(GetY() + GetHeight()/2 < mPos.GetY()) {
173                             SetDirSafe(MmgDir.DIR_FRONT);
174                         } else if(GetY() > mPos.GetY() +
                            GetHeight()/2) {
175                             SetDirSafe(MmgDir.DIR_BACK);
176                         } else if(GetX() + GetWidth()/2 < mPos.
                            GetX()) {
177                             SetDirSafe(MmgDir.DIR_RIGHT);
178                         } else {
179                             SetDirSafe(MmgDir.DIR_LEFT);
180                         }
181                     }
182                 } else {
183                     isMoving = false;
184                 }
185             }
186
187             if(dir != MmgDir.DIR_NONE) {
188                 current = new MmgRect(subj.GetX(), subj.GetY(),
                    subj.GetY() + subj.GetHeight(), subj.GetX() + subj.
                    GetWidth());
189                 if(speed < 0) {
190                     speed *= -1;
191                 }
192
193                 if(isMoving == true) {
194                     if(dir == MmgDir.DIR_BACK) {
195                         if(subj.GetY() - speed >= ScreenGame.
                            BOARD_TOP) {
196                             current.ShiftRect(0, (speed * -1));
197                             coll = screen.CanMove(current, this);
198                             if(coll == null) {
199                                 SetY(current.GetTop());
200                             } else {
201                                 screen.UpdateProcessCollision
                                    (this, coll);
```

```
                                    if(playerType == MdtPlayerType.
                                    ENEMY) {
                                        motorMoveMs =
                                        motorMoveLengthMs;
                                    }
                                }
                            } else {
                                SetY(ScreenGame.BOARD_TOP);
                            }
                        } else if(dir == MmgDir.DIR_FRONT) {
                            if(subj.GetY() + subj.GetHeight() + speed
                            <= ScreenGame.BOARD_BOTTOM) {
                                current.ShiftRect(0, (speed * 1));
                                coll = screen.CanMove(current, this);
                                if(coll == null) {
                                    SetY(current.GetTop());
                                } else {
                                    screen.UpdateProcessCollision
                                    (this, coll);
                                     if(playerType == MdtPlayerType.ENEMY) {
                                        motorMoveMs =
                                        motorMoveLengthMs;
                                    }
                                }
                            } else {
                                SetY(ScreenGame.BOARD_BOTTOM - subj.
                                GetHeight());
                            }
                        } else if(dir == MmgDir.DIR_LEFT) {
                            if(subj.GetX() - speed >= ScreenGame.BOARD_
                            LEFT) {
                                current.ShiftRect((speed * -1), 0);
                                coll = screen.CanMove(current, this);
                                if(coll == null) {
                                    SetX(current.GetLeft());
```

```
                            } else {
                                screen.UpdateProcessCollision(this,
                                coll);
                              if(playerType == MdtPlayerType.ENEMY) {
                                 motorMoveMs = motorMoveLengthMs;
                                }
                            }
                        } else {
                            SetX(ScreenGame.BOARD_LEFT);
                        }
                    } else if(dir == MmgDir.DIR_RIGHT) {
                        if(subj.GetX() + subj.GetWidth() + speed <=
                        ScreenGame.BOARD_RIGHT) {
                            current.ShiftRect((speed * 1), 0);
                            coll = screen.CanMove(current, this);
                            if(coll == null) {
                                SetX(current.GetLeft());
                            } else {
                                screen.UpdateProcessCollision(this,
                                coll);
                                if(playerType == MdtPlayerType.
                                ENEMY) {
                                    motorMoveMs = motorMoveLengthMs;
                                }
                            }
                        } else {
                            SetX(ScreenGame.BOARD_RIGHT - subj.
                            GetWidth());
                        }
                    }
                }
            }
        } else {
            if(weaponCurrent.attackType == MdtWeaponAttackType.
            THROWING) {
```

```
259                     if(weaponCurrent.screen == null) {
260                         weaponCurrent.MmgUpdate(updateTick,
                            currentTimeMs, msSinceLastFrame);
261                     }
262
263                     if(weaponCurrent.throwingTimeMsCurrent >=
                        weaponCurrent.throwingCoolDown) {
264                         isAttacking = false;
265                     }
266                 } else if(weaponCurrent.attackType ==
                    MdtWeaponAttackType.STABBING) {
267                     if(weaponCurrent.screen == null) {
268                         weaponCurrent.MmgUpdate(updateTick,
                            currentTimeMs, msSinceLastFrame);
269                     }
270
271                     if(weaponCurrent.animTimeMsCurrent >
                        (weaponCurrent.animTimeMsTotal + weaponCurrent.
                        stabbingCoolDown)) {
272                         isAttacking = false;
273                     }
274                 }
275
276                 current = weaponCurrent.GetWeaponRect();
277                 if(current != null) {
278                     coll = screen.CanMove(current, weaponCurrent);
279                     if(coll != null) {
280                         screen.UpdateProcessWeaponCollision(coll, this,
                            current);
281                     }
282                 }
283             }
284         }
285     }
286     return lret;
287 }
```

```
@Override
001 public void MmgDraw(MmgPen p) {
002     if (isVisible == true) {
003         if(isBroken) {
004             subjBreaks.MmgDraw(p);
005         } else {
006             if(isAttacking) {
007                 if(dir == MmgDir.DIR_BACK) {
008                     weaponCurrent.MmgDraw(p);
009                     subj.MmgDraw(p);
010                 } else if(dir == MmgDir.DIR_FRONT) {
011                     subj.MmgDraw(p);
012                     weaponCurrent.MmgDraw(p);
013                 } else if(dir == MmgDir.DIR_LEFT) {
014                     weaponCurrent.MmgDraw(p);
015                     subj.MmgDraw(p);
016                 } else if(dir == MmgDir.DIR_RIGHT) {
017                     weaponCurrent.MmgDraw(p);
018                     subj.MmgDraw(p);
019                 }
020             } else {
021                 subj.MmgDraw(p);
022             }
023         }
024     }
025 }
```

Looking at the `MmgUpdate` method first, the section of code from lines 5 to 11 is responsible for running through the `subjBreak` animation frames once the character has been defeated. On the last frame of the animation, the character requests to be removed from the current game screen, line 11. The next few lines of code, lines 13–17, check to see if the current character is broken or not as well as update the `subj` class field.

If the current character is being bounced, then the code on lines 20–122 is executed. An `MmgRect` is created to represent the character's current position and dimensions, line 23. The value of the speed field is checked and adjusted to be positive if need be. New X and Y values, used to position this character, are initialized on lines 28 and 29.

The values for the method's `nx` and `ny` variables are calculated based on the speed and direction of the character on lines 31–41.

Based on the four directions the character is moved, a bounce can be triggered if the current character collides with another character, player or enemy, on lines 43–117. When the duration of the bounce is up, the feature is deactivated and reset on lines 119 and 120. The block of code from lines 124 to 184 is used to process moving an enemy character around the board. Collisions and subsequent interactions are handled by the `ScreenGame` class' `UpdateProcessCollision` method, which we'll review in detail soon. Also, it's in the `UpdateProcessCollision` method that item collisions and object pushing are handled.

The block of code on lines 187–256 is responsible for processing the character's movement whether or not it comes from a user's input or from an enemy's AI calculations. This is an important part of the game's code because the collision handling performed here drives a lot of the game's specifications. For instance, items, objects, weapons, and movement interactions are all handled through the collision detection process.

The last block of code on lines 258–282 is used to animate the character's weapon if attacking. A collision check is performed on the weapon resulting in a call to the `ScreenGame` class' `UpdateProcessWeaponCollision` method. This is the method that will handle weapon strikes performed by players.

The last method for us to review is the `MmgDraw` method. Take a quick look at the method listed in the preceding. Notice that the method supports drawing the subject's break animation or an attack animation with the weapon properly positioned above or below the character. Lastly, the character can be drawn without a weapon. Quite a lot going on in a few lines of code.

Take a moment to process everything. Update your copy of the `MdtCharInter` class and add all of the support and main methods we've reviewed here. If you're following along in C#, be sure to catch any syntax differences between the two languages. Feel free to copy and paste the entire class from the completed chapter project contained in the game engine project. If you do so, make sure to update the `package` or `namespace` appropriately.

MdtItemPotion: Class Review

The MdtItemPotion class is a level 1 extended class, extending directly from the MdtItem base class. It's also a new base class as it's used as the basis for the three potion items available in the game. The MdtItemPotion class doesn't have any static class members or enumerations we need to cover, and given its simplicity, we can list the entire class here taking into account adjustments necessary for the C# version.

Listing 23-10. MdtItemPotion Class Review 1

```
//Java Version
public class MdtItemPotion extends MdtItem {

//C# Version
public class MdtItemPotion : MdtItem {

public MdtItemPotionType potionType = MdtItemPotionType.NONE;

//Java Version
01 public MdtItemPotion(MmgBmp Subj, MdtItemPotionType PotionType,
   MdtPointsType Points) {
02      super(Subj, MdtObjType.ITEM, MdtObjSubType.ITEM_POTION, Points);

//C# Version
01 public MdtItemPotion(MmgBmp Subj, MdtItemPotionType PotionType,
   MdtPointsType Points)
02      : base(Subj, MdtObjType.ITEM, MdtObjSubType.ITEM_POTION, Points) {

03       SetPotionType(PotionType);
04 }

//Java Version
@Override
01 public MmgBmp GetSubj() {

//C# Version
01 public override MmgBmp GetSubj() {

02     return (MmgBmp)subj;
03 }
```

```
01 public MdtItemPotionType GetPotionType() {
02      return potionType;
03 }

01 public void SetPotionType(MdtItemPotionType PotionType) {
02      potionType = PotionType;
03 }
}
```

The MdtItemPotion class adds MdtItemPotionType support to the MdtItem class. As you can see from the code listed in the preceding, a new class field with associated get and set methods has been added to the class. The class constructor has been adjusted from the signature found in the MdtItem class to support a potion type argument. Also note that the base class constructor is called with an object type and subtype defined by default.

The new class field is initialized on line 3 of the class constructor. The only other thing I should mention here is that the GetSubj method is overridden to return an MmgBmp object instance instead of the broader MmgObj instance returned by the base, MdtItem, class. This is normal behavior for extended classes. Make sure to add this class to your DungeonTrap project. Next up, we'll check out a level 1 extended class that's a direct implementation of the super/base class.

MdtItemCoinBag: Class Review

The MdtItemCoinBag class is a level 1 extended class, extending directly from the MdtItem class. Unlike the MdtItemPotion class however, the coin bag class doesn't add any new functionality to the MdtItem class. It simply defines a more concrete item class with the subject image, object type, and object subtype defined. This class is also very concise, so I'll list the contents of the class here for both the Java and C# versions of the game.

Listing 23-11. MdtItemCoinBag Class Review 1

```
//Java Version
public class MdtItemCoinBag extends MdtItem {

//C# Version
public class MdtItemCoinBag : MdtItem {
```

```
//Java Version
01 public MdtItemCoinBag() {
02      super(MmgBmpScaler.ScaleMmgBmp(MmgHelper. GetBasicCachedBmp("bag_
        coins_lg.png"), 1.5d, true), MdtObjType.ITEM, MdtObjSubType.ITEM_
        COINS, MdtPointsType.PTS_1000);

//C# Version
01 public MdtItemCoinBag()
02       : base(MmgBmpScaler.ScaleMmgBmp(MmgHelper. GetBasicCachedBmp("bag_
         coins_lg.png"), 1.5d, true), MdtObjType.ITEM, MdtObjSubType.ITEM_
         COINS, MdtPointsType.PTS_1000) {

03 }

//Java Version
01 public MdtItemCoinBag(MmgBmp Subj) {
02         super(Subj, MdtObjType.ITEM, MdtObjSubType.ITEM_COINS,
           MdtPointsType.PTS_1000);

03 }
}
```

As you can see from the class code listed in the preceding, the `MdtItemCoinBag` class is simply a specific implementation of the `MdtItem` base class. Make sure to add this class to your DungeonTrap game project. As always, you can check the chapter's completed code in the game engine's project folder is you run into any problems.

MdtObjPush: Class Review

The `MdtObjPush` class is a level 1 extended class because it directly extends the `MdtObj` base class. It's also a new base class because there are three level 2 classes that extend from the `MdtObjPush` class. This class adds the ability for an object to be pushed by player characters. The class also supports displaying a breaking animation similar to the way that the `MdtCharInter` class supports it.

MdtObjPush: Class Fields

The MdtObjPush class has no static class members or enumerations for us to review, so we've begun the review process by looking at the class' fields.

Listing 23-12. MdtObjPush Class Fields 1

```
public boolean breakOnFirst = true;
public MmgSprite subjBreaks = null;
public boolean isBroken = false;
private boolean lret = false;
```

The first block of class fields for us to review with regard to the MdtObjPush class are listed in the preceding. The breakOnFirst field is a Boolean flag that indicates if the object should break on the first object that it collides with, after being pushed. Currently, setting this to true is the default functionality. The next class field listed in the preceding group is the subjBreaks field, which is an instance of the MmgSprite class and is used to play an animation when the object is broken. A Boolean flag, isBroken, is used to indicate if the current object has been broken in a collision. The lret field is also a Boolean flag, but this field is used internally by certain class methods.

Listing 23-13. MdtObjPush Class Fields 2

```
public boolean isBeingPushed = false;
public int pushDir = MmgDir.DIR_NONE;
public int pushSpeed = ScreenGame.GetSpeedPerFrame(280);
public MdtPlayerType pushedBy;
```

The next block of class fields, listed in the preceding, are used to support the object's push feature. The first field listed, isBeingPushed, is used to indicate if the current object is being pushed or not. The subsequent field, pushDir, is used to indicate in which direction the object is being pushed. The last two fields represent the speed, pushSpeed, that the current object travels at when pushed and, if set, the type of player that pushed the object, pushedBy.

Listing 23-14. MdtObjPush Class Fields 3

```
public MmgRect current = null;
public MdtBase coll = null;
public ScreenGame screen = null;
```

The last group of class fields are used in the collision detection process. The current field is an instance of the `MmgRect` class, and it represents the current position and dimensions of the pushed object. The next entry in the group is an instance of the `MdtBase` class, and it's used to mark a collision with another game object. The last entry is the `screen` field. This field is used to reference the game screen that the current object belongs to. That wraps up the class' field review. Next, we'll take a look at the class method outline and definitions.

Listing 23-15. MdtObjPush Pertinent Method Outline 1

```
//Main Methods
public MdtObjPush() { ... }
public MdtObjPush(MmgBmp Subj, MmgSprite SubjBreaks, MdtObjType ObjType,
MdtObjSubType ObjSubType, ScreenGame Screen) { ... }

public MmgBmp GetSubj() { ... }
public void SetPosition(MmgVector2 v) { ... }
public void SetPosition(int x, int y) { ... }
public void SetX(int i) { ... }
public void SetY(int i) { ... }
public boolean MmgUpdate(int updateTick, long currentTimeMs, long
msSinceLastFrame) { ... }

public void MmgDraw(MmgPen p) { ... }

//Support Methods
public boolean GetBreakOnFirst() { ... }
public void SetBreakOnFirst(boolean b) { ... }
public MmgSprite GetSubjBreaks() { ... }
public void SetSubjBreaks(MmgSprite s) { ... }
public boolean GetIsBroken() { ... }
public void SetIsBroken(boolean b) { ... }
public boolean GetIsBeingPushed() { ... }
```

```
public void SetIsBeingPushed(boolean b) { ... }
public int GetPushDir() { ... }
public void SetPushDir(int i) { ... }
public int GetPushSpeed() { ... }
public void SetPushSpeed(int i) { ... }
public MdtPlayerType GetPushedBy() { ... }
public void SetPushedBy(MdtPlayerType p) { ... }
public ScreenGame GetScreen() { ... }
public void SetScreen(ScreenGame o) { ... }
```

The class definition for the `MdtObjPush` class is listed in the following for both Java and C# versions of the class.

Listing 23-16. MdtObjPush Class Definitions 1

```
//Java Version
public class MdtObjPush extends MdtObj {

//C# Version
public class MdtObjPush : MdtObj {
```

As always, if you're following along in C#, be aware of syntax differences between Java and C#. Don't forget you can verify your code against the chapter's completed project code contained in the game engine's project directory.

MdtObjPush: Support Method Details

The `MdtObjPush` class' support methods are basic get and set methods. We'll list all the methods here, but we won't go over them in any detail.

Listing 23-17. MdtObjPush Support Method Details 1

```
01 public boolean GetBreakOnFirst() {
02     return breakOnFirst;
03 }

01 public void SetBreakOnFirst(boolean b) {
02     breakOnFirst = b;
03 }
```

```
01 public MmgSprite GetSubjBreaks() {
02     return subjBreaks;
03 }

01 public void SetSubjBreaks(MmgSprite s) {
02     subjBreaks = s;
03 }

01 public boolean GetIsBroken() {
02     return isBroken;
03 }

01 public void SetIsBroken(boolean b) {
02     isBroken = b;
03 }

01 public boolean GetIsBeingPushed() {
02     return isBeingPushed;
03 }

01 public void SetIsBeingPushed(boolean b) {
02     isBeingPushed = b;
03 }

01 public int GetPushDir() {
02     return pushDir;
03 }

01 public void SetPushDir(int i) {
02     pushDir = i;
03 }

01 public int GetPushSpeed() {
02     return pushSpeed;
03 }

01 public void SetPushSpeed(int i) {
02     pushSpeed = i;
03 }
```

```
01 public MdtPlayerType GetPushedBy() {
02     return pushedBy;
03 }

01 public void SetPushedBy(MdtPlayerType p) {
02     pushedBy = p;
03 }

01 public ScreenGame GetScreen() {
02     return screen;
03 }

01 public void SetScreen(ScreenGame o) {
02     screen = o;
03 }
```

As always, you can add the methods to your copy of the `MdtObjPush` class, or you can copy and paste the entire class into your project alongside the existing classes.

MdtObjPush: Main Method Details

In this section, we'll be reviewing in detail the `MdtObjPush` class' main methods. I've broken down the main methods into three groups. The first such group is listed in the following.

Listing 23-18. MdtObjPush Main Method Details 1

```
01 public MdtObjPush() {
02
03 }

01 public MdtObjPush(MmgBmp Subj, MmgSprite SubjBreaks, MdtObjType ObjType,
   MdtObjSubType ObjSubType, ScreenGame Screen) {
02     super(Subj, ObjType, ObjSubType);
03     SetScreen(Screen);
04     SetSubjBreaks(SubjBreaks);
05 }

@Override
```

```
01 public MmgBmp GetSubj() {
02     return (MmgBmp)subj;
03 }
```

The first set of main methods listed in the preceding starts with the two class constructors. The first constructor takes no arguments and doesn't configure any class fields. The second constructor listed is an advanced constructor that takes arguments whose values are used to configure the class fields. The last method listed in this set is a get method for the `subj` class field. Notice how the `subj` field is cast down from an `MmgObj` to an `MmgBmp` object instance to bring the method in line with the class' functionality. The next set of main methods should look familiar.

Listing 23-19. MdtObjPush Main Method Details 2

```
@Override
01 public void SetPosition(MmgVector2 v) {
02     super.SetPosition(v);
03     subj.SetPosition(v);
04     subjBreaks.SetPosition(v.GetX() + (subj.GetWidth() - subjBreaks.
       GetWidth())/2, v.GetY() + (subj.GetHeight() - subjBreaks.
       GetHeight())/2);
05 }

@Override
01 public void SetPosition(int x, int y) {
02     super.SetPosition(x, y);
03     subj.SetPosition(x, y);
04     subjBreaks.SetPosition(x + (subj.GetWidth() - subjBreaks.
       GetWidth())/2, y + (subj.GetHeight() - subjBreaks.GetHeight())/2);
05 }

@Override
01 public void SetX(int i) {
02     super.SetX(i);
03     subj.SetX(i);
04     subjBreaks.SetX(i + (subj.GetWidth() - subjBreaks.GetWidth())/2);
05 }
```

```
@Override
01 public void SetY(int i) {
02     super.SetY(i);
03     subj.SetY(i);
04     subjBreaks.SetY(i + (subj.GetHeight() - subjBreaks.GetHeight())/2);
05 }
```

The second set of main methods listed in the preceding demonstrates overriding methods to keep the object, the subject, and the subject break animation all properly positioned. We've seen this pattern before during the `MdtItem` and `MdtObj` class reviews. The next two main methods to review are the drawing routine methods `MmgUpdate` and `MmgDraw`.

Listing 23-20. MdtObjPush Main Method Details 3

```
@Override
001 public boolean MmgUpdate(int updateTick, long currentTimeMs, long
    msSinceLastFrame) {
002     lret = false;
003     if (isVisible == true) {
004         if(isBroken) {
005             subjBreaks.MmgUpdate(updateTick, currentTimeMs,
                msSinceLastFrame);
006             if(subjBreaks.GetFrameIdx() == subjBreaks.GetFrameStop()) {
007                 screen.UpdateRemoveObj(this, pushedBy);
008             }
009         } else {
010             subj.MmgUpdate(updateTick, currentTimeMs, msSinceLastFrame);
011
012             if(pushDir != MmgDir.DIR_NONE) {
013                 current = new MmgRect(subj.GetX(), subj.GetY(),
                    subj.GetY() + subj.GetHeight(), subj.GetX() + subj.
                    GetWidth());
014                 if(pushSpeed < 0) {
015                     pushSpeed *= -1;
016                 }
017
018                 if(isBeingPushed == true) {
```

```
019                     if(pushDir == MmgDir.DIR_BACK) {
020                         if(subj.GetY() - pushSpeed >= ScreenGame.BOARD_
                            TOP) {
021                             current.ShiftRect(0, (pushSpeed * -1));
022                             coll = screen.CanMove(current, this);
023                             if(coll == null) {
024                                 SetY(current.GetTop());
025                             } else {
026                                 if(coll.GetMdtType() == MdtObjType.
                                    PLAYER) {
027                                     ((MdtCharInter)coll).
                                        Bounce(GetPosition().Clone(),
                                        GetWidth()/2, GetHeight()/2,
                                        pushDir, pushedBy);
028                                 } else if(coll.GetMdtType() ==
                                    MdtObjType.ENEMY) {
029                                     ((MdtCharInter)coll).
                                        Bounce(GetPosition().Clone(),
                                        GetWidth()/2, GetHeight()/2,
                                        pushDir, pushedBy);
030                                 }
031
032                                 if(coll.GetMdtType() == MdtObjType.
                                    ENEMY || coll.GetMdtType() ==
                                    MdtObjType.PLAYER || coll.GetMdtType()
                                    == MdtObjType.OBJECT) {
033                                     if(breakOnFirst) {
034                                         isBeingPushed = false;
035                                         pushDir = MmgDir.DIR_NONE;
036                                         isBroken = true;
037                                     } else {
038                                         SetY(current.GetTop());
039                                     }
040                                 } else {
041                                     SetY(current.GetTop());
042                                 }
```

```
043                             }
044                         } else {
045                             SetY(ScreenGame.BOARD_TOP);
046                             isBeingPushed = false;
047                             pushDir = MmgDir.DIR_NONE;
048                             isBroken = true;
049                         }
050                     } else if(pushDir == MmgDir.DIR_FRONT) {
051                         if(subj.GetY() + subj.GetHeight() + pushSpeed
                            <= ScreenGame.BOARD_BOTTOM) {
052                             current.ShiftRect(0, (pushSpeed * 1));
053                             coll = screen.CanMove(current, this);
054                             if(coll == null) {
055                                 SetY(current.GetTop());
056                             } else {
057                                 if(coll.GetMdtType() == MdtObjType.
                                    PLAYER) {
058                                     ((MdtCharInter)coll).
                                        Bounce(GetPosition().Clone(),
                                        GetWidth()/2, GetHeight()/2,
                                        pushDir, pushedBy);
059                                 } else if(coll.GetMdtType() ==
                                    MdtObjType.ENEMY) {
060                                     ((MdtCharInter)coll).
                                        Bounce(GetPosition().Clone(),
                                        GetWidth()/2, GetHeight()/2,
                                        pushDir, pushedBy);
061                                 }
062
063                                 if(coll.GetMdtType() == MdtObjType.
                                    ENEMY || coll.GetMdtType() ==
                                    MdtObjType.PLAYER || coll.GetMdtType()
                                    == MdtObjType.OBJECT) {
064                                     if(breakOnFirst) {
065                                         isBeingPushed = false;
066                                         pushDir = MmgDir.DIR_NONE;
```

```
067                                 isBroken = true;
068                             } else {
069                                 SetY(current.GetTop());
070                             }
071                         } else {
072                             SetY(current.GetTop());
073                         }
074                     }
075                 } else {
076                     SetY(ScreenGame.BOARD_BOTTOM - subj.
                    GetHeight());
077                     isBeingPushed = false;
078                     pushDir = MmgDir.DIR_NONE;
079                     isBroken = true;
080                 }
081             } else if(pushDir == MmgDir.DIR_LEFT) {
082                 if(subj.GetX() - pushSpeed >= ScreenGame.BOARD_
                LEFT) {
083                     current.ShiftRect((pushSpeed * -1), 0);
084                     coll = screen.CanMove(current, this);
085                     if(coll == null) {
086                         SetX(current.GetLeft());
087                     } else {
088                         if(coll.GetMdtType() == MdtObjType.
                        PLAYER) {
089                             ((MdtCharInter)coll).
                            Bounce(GetPosition().Clone(),
                            GetWidth()/2, GetHeight()/2,
                            pushDir, pushedBy);
090                         } else if(coll.GetMdtType() ==
                        MdtObjType.ENEMY) {
091                             ((MdtCharInter)coll).
                            Bounce(GetPosition().Clone(),
                            GetWidth()/2, GetHeight()/2,
                            pushDir, pushedBy);
092                         }
```

```
093
094                                 if(coll.GetMdtType() == MdtObjType.
                                    ENEMY || coll.GetMdtType() ==
                                    MdtObjType.PLAYER || coll.GetMdtType()
                                    == MdtObjType.OBJECT) {
095                                     if(breakOnFirst) {
096                                         isBeingPushed = false;
097                                         pushDir = MmgDir.DIR_NONE;
098                                         isBroken = true;
099                                     } else {
100                                         SetX(current.GetLeft());
101                                     }
102                                 } else {
103                                     SetX(current.GetLeft());
104                                 }
105                             }
106                         } else {
107                             SetX(ScreenGame.BOARD_LEFT);
108                             isBeingPushed = false;
109                             pushDir = MmgDir.DIR_NONE;
110                             isBroken = true;
111                         }
112                     } else if(pushDir == MmgDir.DIR_RIGHT) {
113                         if(subj.GetX() + subj.GetWidth() + pushSpeed <=
                            ScreenGame.BOARD_RIGHT) {
114                             current.ShiftRect((pushSpeed * 1), 0);
115                             coll = screen.CanMove(current, this);
116                             if(coll == null) {
117                                 SetX(current.GetLeft());
118                             } else {
119                                 if(coll.GetMdtType() == MdtObjType.
                                    PLAYER) {
120                                     ((MdtCharInter)coll).
                                     Bounce(GetPosition().Clone(),
                                     GetWidth()/2, GetHeight()/2,
                                     pushDir, pushedBy);
```

```
121                             } else if(coll.GetMdtType() ==
                                MdtObjType.ENEMY) {
122                                 ((MdtCharInter)coll).
                                    Bounce(GetPosition().Clone(),
                                    GetWidth()/2, GetHeight()/2,
                                    pushDir, pushedBy);
123                             }
124
125                             if(coll.GetMdtType() == MdtObjType.
                                ENEMY || coll.GetMdtType() ==
                                MdtObjType.PLAYER || coll.GetMdtType()
                                == MdtObjType.OBJECT) {
126                                 if(breakOnFirst) {
127                                     isBeingPushed = false;
128                                     pushDir = MmgDir.DIR_NONE;
129                                     isBroken = true;
130                                 } else {
131                                     SetX(current.GetLeft());
132                                 }
133                             } else {
134                                 SetX(current.GetLeft());
135                             }
136                         }
137                     } else {
138                       SetX(ScreenGame.BOARD_RIGHT - subj.GetWidth());
139                         isBeingPushed = false;
140                         pushDir = MmgDir.DIR_NONE;
141                         isBroken = true;
142                     }
143                 }
144             }
145         }
146     }
147     lret = true;
148 }
```

```
149     return lret;
150 }

@Override
001 public void MmgDraw(MmgPen p) {
002     if (isVisible == true) {
003         if(isBroken) {
004             subjBreaks.MmgDraw(p);
005         } else {
006             subj.MmgDraw(p);
007         }
009     }
010 }
```

The first method listed is the `MmgUpdate` method. It is a very long method, so if you don't want to type up the class while you review it, I'll provide instructions on how to copy and paste in completed level 1 classes at the end of their review. This method is responsible for animating the object in the direction of the push. On lines 5–8, the subject break animation is updated. Of course, this only occurs when the current object is marked as broken.

When the animation reaches the last frame, the object requests to be removed from the game screen with a call to the `UpdateRemoveObj` method of the `ScreenGame` class. The `subj` field has its `MmgUpdate` method called on line 10. If the push direction has been set, then the current `MmgRect` field is updated, and the value of the `pushSpeed` field is validated and forced to be positive, lines 13–16. The main block of code runs from line 18 to line 144.

If the object is being pushed and depending on the direction the object is being pushed in, collisions and positioning are handled. In each case, if the object reaches the edge of the board, the object is broken, and the resulting break animation is triggered. The last method to review in the set listed in the preceding is the `MmgDraw` method. Notice that the method can draw the subject break animation or the normal subject depending on the object's internal state.

MdtObjTorch: Class Review

The `MdtObjTorch` class is a level 1 extended class, but it doesn't establish a new base class as we've seen with the `MdtItemPotion` class. The `MdtObjTorch` class extends the `MdtObj` class and provides some slightly new functionality. Essentially, the `MdtObjTorch`

class adds the ability to have the torch be in an on or off state. This class has no static class members or enumerations for us to review, so we'll start the process in the class fields section. Let's take a look at some code!

MdtObjTorch: Class Fields

The `MdtObjTorch` class adds some new functionality to the `MdtObj` base class. Let's review the class' fields and see what we've got.

Listing 23-21. MdtObjTorch Class Fields 1

```
public MmgBmp subjOff = null;
public boolean isBurning = false;
private boolean lret = false;
```

There are two public class fields listed in the preceding. The `subjOff` class field is an `MmgBmp` instance used to display the torch when the flame is off. To enable this functionality, the `isBurning` class field is set to false. The last field listed in the set is a private class field used internally by certain class methods. Next up, we'll take a look at the class method outline and definitions.

MdtObjTorch: Pertinent Method Outline

The `MdtObjTorch` class' method outline is listed in the following with methods separated into two groups, main and support methods.

Listing 23-22. MdtObjTorch Pertinent Method Outline 1

```
//Main Methods
public MdtObjTorch() { ... }
public MdtObjTorch(MmgSprite Subj, MmgBmp SubjOff) { ... }

public void SetPosition(MmgVector2 v) { ... }
public void SetPosition(int x, int y) { ... }
public void SetX(int i) { ... }
public void SetY(int i) { ... }
public boolean MmgUpdate(int updateTick, long currentTimeMs, long
msSinceLastFrame) { ... }
```

```
public void MmgDraw(MmgPen p) { ... }

//Support Methods
public MmgBmp GetSubjOff() { ... }
public void SetSubjOff(MmgBmp SubjOff) { ... }
public boolean GetIsBurning() { ... }
public void SetIsBurning(boolean b) { ... }
```

I'd also like to take a moment to review the class definition in both Java and C#, listed in the following.

Listing 23-23. MdtObjTorch Class Definitions 1

```
//Java Version
public class MdtObjTorch extends MdtObj {

//C# Version
public class MdtObjTorch : MdtObj {
```

Make sure that you are aware of any syntax differences between C# and Java if you're following along in C#. Always look at the current class we're reviewing in the completed chapter project; note things like `import/using` statements and key syntax changes. In the next section, we'll take a look at the class' support methods.

MdtObjTorch: Support Method Details

The `MdtObjTorch` class introduces four new support methods to the class. These are simple get and set methods that provide access to the class fields.

Listing 23-24. MdtObjTorch Support Method Details 1

```
01 public MmgBmp GetSubjOff() {
02     return subjOff;
03 }

01 public void SetSubjOff(MmgBmp SubjOff) {
02     subjOff = SubjOff;
03 }
```

```
01 public boolean GetIsBurning() {
02     return isBurning;
03 }
```

```
01 public void SetIsBurning(boolean b) {
02     isBurning = b;
03 }
```

Be sure to review and understand the code before adding it to your copy of the MdtObjTorch file. Again, I'll provide instructions on how to add all the level 1 class files to your project in one step, in just a bit.

MdtObjTorch: Main Method Details

We'll begin the main method review section by covering the class constructors and the overridden GetSubj method.

Listing 23-25. MdtObjTorch Main Method Details 1

```
01 public MdtObjTorch() {
02     MmgBmp src = MmgHelper.GetBasicCachedBmp("torch_spritesheet_
       lg.png");
03     MmgSpriteSheet ssSrc = new MmgSpriteSheet(src, 32, 32);
04     MmgSprite subj = new MmgSprite(ssSrc.GetFrames());
05     subj.SetMsPerFrame(280);
06     SetSubj(subj);
07     SetSubjOff(MmgHelper.GetBasicCachedBmp("torch_off.png"));
08     SetMdtType(MdtObjType.OBJECT);
09     SetMdtSubType(MdtObjSubType.OBJECT_TORCH);
10     SetWidth(subj.GetWidth());
11     SetHeight(subj.GetHeight());
12 }
```

```
01 public MdtObjTorch(MmgSprite Subj, MmgBmp SubjOff) {
02     super(Subj, MdtObjType.OBJECT, MdtObjSubType.OBJECT_TORCH);
03     GetSubj().SetMsPerFrame(280);
04     SetSubjOff(SubjOff);
05 }
```

```
@Override
01 public MmgSprite GetSubj() {
02     return (MmgSprite)subj;
03 }
```

The first constructor listed in this block takes no arguments and loads the resources necessary to drive a torch's default functionality. The `subj` field is set to a flickering torch animation, lines 2–6. Similarly, the `subjOff` field is set based on the "torch_off" resource, line 7. The remaining lines of code set default class field values. The next entry in the block of main methods is a constructor that takes an `MmgSprite` and an `MmgBmp` object as arguments.

The base class constructor is called on line 2, and the subject off field is initialized on line 4. The last main method listed in this block is the overridden `GetSubj` method. This implementation forces the `subj` field to be returned as an `MmgSprite` instance. This is a more concrete implementation of the base class' `GetSubj` method, which returns an `MmgObj` instance.

Listing 23-26. MdtObjTorch Main Method Details 2

```
@Override
01 public void SetPosition(MmgVector2 v) {
02     super.SetPosition(v);
03     subj.SetPosition(v);
04     subjOff.SetPosition(v);
05 }

@Override
01 public void SetPosition(int x, int y) {
02     super.SetPosition(x, y);
03     subj.SetPosition(x, y);
04     subjOff.SetPosition(x, y);
05 }

@Override
01 public void SetX(int i) {
02     super.SetX(i);
03     subj.SetX(i);
04     subjOff.SetX(i);
05 }
```

```
@Override
01 public void SetY(int i) {
02     super.SetY(i);
03     subj.SetY(i);
04     subjOff.SetY(i);
05 }
```

The next block of main methods is a pattern we should be familiar with. The positioning methods are overridden to ensure both the `subj` and `subjOff` objects are properly aligned with each other.

Listing 23-27. MdtObjTorch Main Method Details 3

```
@Override
01 public boolean MmgUpdate(int updateTick, long currentTimeMs, long
   msSinceLastFrame) {
02     lret = false;
03     if (isVisible == true) {
04         subj.MmgUpdate(updateTick, currentTimeMs, msSinceLastFrame);
05         lret = true;
06     }
07     return lret;
08 }

@Override
01 public void MmgDraw(MmgPen p) {
02     if (isVisible == true) {
03         if(isBurning) {
04             subj.MmgDraw(p);
05         } else {
06             subjOff.MmgDraw(p);
07         }
08     }
09 }
```

The last block of main methods to review are the game engine drawing routine methods `MmgUpdate` and `MmgDraw`. Note the simplicity of this extended class implementation. The torch animation is powered by the `subj` class field updated in

the MmgUpdate method. The MmgDraw method detects if the torch is off and displays the appropriate image. That wraps up our review of the MdtObjTorch class. Make sure your copy of the class matches the expected functionality in the chapter's completed project folder contained in the game engine project.

MdtWeaponSpear: Class Review

The MdtWeaponSpear class is similar to the MdtObjTorch class we just finished reviewing. They are both level 1 extended classes that extend a base class and are not extended themselves. In this case, the base class is the MdtWeapon class. We're not really adding any new functionality here. We're just expressing existing functionality more concretely. The class is concise, so we can list the entire thing right here for both Java and C#.

Listing 23-28. MdtWeaponSpear Class Review 1

```
//Java Version
public class MdtWeaponSpear extends MdtWeapon {

//C# Version
public class MdtWeaponSpear : MdtWeapon {

    01 public MdtWeaponSpear(MdtChar Holder, MdtWeaponType WeaponType,
       MdtPlayerType Player) {
    02     super(Holder, WeaponType, Player);
    03     subjFront = MmgHelper.GetBasicCachedBmp("weapon_spear_dir_front.png");
    04     subjFront = MmgBmpScaler.ScaleMmgBmp(subjFront, 2.0f, true);
    05
    06     subjBack = MmgHelper.GetBasicCachedBmp("weapon_spear_dir_back.png");
    07     subjBack = MmgBmpScaler.ScaleMmgBmp(subjBack, 2.0f, true);
    08
    09     subjLeft = MmgHelper.GetBasicCachedBmp("weapon_spear_dir_left.png");
    10     subjLeft = MmgBmpScaler.ScaleMmgBmp(subjLeft, 2.0f, true);
    11
    12     subjRight = MmgHelper.GetBasicCachedBmp("weapon_spear_dir_right.png");
    13     subjRight = MmgBmpScaler.ScaleMmgBmp(subjRight, 2.0f, true);
    14
    15     SetMdtType(MdtObjType.WEAPON);
    16     SetMdtSubType(MdtObjSubType.WEAPON_SPEAR);
```

```
17      SetAttackType(MdtWeaponAttackType.STABBING);
18      SetWidth(subjBack.GetHeight());
19      SetHeight(subjBack.GetHeight());
20      SetDamage(1);
21      SetAnimTimeMsTotal(250);
22  }

@Override
01  public MdtWeaponSpear Clone() {
02      MdtWeaponSpear ret = new MdtWeaponSpear(holder, weaponType,
        player);
03      ret.SetAnimPrctComplete(GetAnimPrctComplete());
04      ret.SetIsActive(GetIsActive());
05      ret.SetAnimTimeMsCurrent(GetAnimTimeMsCurrent());
06      ret.SetAnimTimeMsTotal(GetAnimTimeMsTotal());
07      ret.SetAttackType(GetAttackType());
08
09      if(GetMmgColor() == null) {
10          ret.SetMmgColor(GetMmgColor());
11      } else {
12          ret.SetMmgColor(GetMmgColor().Clone());
13      }
14
15      if(GetCurrent() == null) {
16          ret.SetCurrent(GetCurrent());
17      } else {
18          ret.SetCurrent(GetCurrent().Clone());
19      }
20
21      ret.SetDamage(GetDamage());
22      ret.SetHeight(GetHeight());
23      ret.SetHasParent(GetHasParent());
24      ret.SetIsVisible(GetIsVisible());
25      ret.SetId(GetId());
26      ret.SetHolder(GetHolder());
27      ret.SetParent(GetParent());
```

```
28
29      if(GetPosition() == null) {
30          ret.SetPosition(GetPosition());
31      } else {
32          ret.SetPosition(GetPosition().Clone());
33      }
34
35      if(subjBack == null) {
36          ret.subjBack = subjBack;
37      } else {
38          ret.subjBack = subjBack.CloneTyped();
39      }
40
41      if(subjFront == null) {
42          ret.subjFront = subjFront;
43      } else {
44          ret.subjFront = subjFront.CloneTyped();
45      }
46
47      if(subjLeft == null) {
48          ret.subjLeft = subjLeft;
49      } else {
50          ret.subjLeft = subjLeft.CloneTyped();
51      }
52
53      if(subjRight == null) {
54          ret.subjRight = subjRight;
55      } else {
56          ret.subjRight = subjRight.CloneTyped();
57      }
58
59      ret.throwingDir = throwingDir;
60      ret.throwingFrame = throwingFrame;
61      ret.throwingPath = throwingPath;
62      ret.throwingSpeed = throwingSpeed;
63      ret.throwingSpeedSkew = throwingSpeedSkew;
```

```
    64      ret.throwingCoolDown = throwingCoolDown;
    65      ret.throwingTimeMsRotation = throwingTimeMsRotation;
    66      ret.throwingTimeMsCurrent = throwingTimeMsCurrent;
    67      ret.screen = screen;
    68      ret.stabbingCoolDown = stabbingCoolDown;
    69      return ret;
    70  }
}
```

Note that the `MdtWeaponSpear` class is simply a concrete implementation of the `MdtWeapon` class. Notice that it loads the correct resources to display the weapon in each direction. The second method listed in the class is the `Clone` method. This method is used when cloning a weapon during a thrown weapon attack. The redefinition of the Clone method is necessary to return the proper class type and allow for future customization of the weapon.

That brings us to the end of the level 1 class review. I've created a folder with a copy of each class reviewed in this section. Open the game engine project's folder and find the "cfg" folder. Locate and open the "asset_src" folder and open the folder named "dungeon_trap_level1_classes." Copy the contents of the directory and paste them into your game project folder alongside the other class files. Double-check the `package` or `namespace` of the newly pasted files to make sure it matches your project's setup.

DungeonTrap: Level 2 Classes

The level 2 classes in DungeonTrap extend from a level 1 base class we've reviewed in the previous section and provide a concrete implementation of the level 1 class. Because of how similar certain level 2 classes are, we'll only review one example from each set of classes. We'll take a look at the `MdtCharInterDemon`, `MdtItemPotionGreen`, and `MdtObjPushBarrel` classes before concluding the section with a review of the `MdtCharInterPlayer` class.

MdtCharInterDemon: Class Review

The `MdtCharInterDemon` class is used to represent the demon enemy type, a concrete implementation of the `MdtCharInter` class. The class is very concise, so we can list the entire class here for both Java and C#.

Listing 23-29. MdtCharInterDemon Class Review 1

```
//Java Version
public class MdtCharInterDemon extends MdtCharInter {

//C# Version
public class MdtCharInterDemon : MdtCharInter {

//Java Version
01 public MdtCharInterDemon(MmgSprite Subj, int FrameFrontS, int
FrameFrontE, int FrameBackS, int FrameBackE, int FrameLeftS, int
FrameLeftE, int FrameRightS, int FrameRightE, ScreenGame Screen) {
     02     super(Subj, FrameFrontS, FrameFrontE, FrameBackS, FrameBackE,
FrameLeftS, FrameLeftE, FrameRightS, FrameRightE, Screen, MdtObjType.ENEMY,
MdtObjSubType.ENEMY_DEMON);

//C# Version
01 public MdtCharInterDemon(MmgSprite Subj, int FrameFrontS, int
FrameFrontE, int FrameBackS, int FrameBackE, int FrameLeftS, int
FrameLeftE, int FrameRightS, int FrameRightE, ScreenGame Screen)
     02      : base(Subj, FrameFrontS, FrameFrontE, FrameBackS, FrameBackE,
FrameLeftS, FrameLeftE, FrameRightS, FrameRightE, Screen, MdtObjType.ENEMY,
MdtObjSubType.ENEMY_DEMON) {

     03    SetPlayerType(MdtPlayerType.ENEMY);
     04    SetHealthMax(2);
     05    SetHealthCurrent(2);
     06    weaponCurrent.SetPlayer(GetPlayerType());
     07    SetMotor(MdtEnemyMotorType.NONE);
     08    SetSpeed(ScreenGame.GetSpeedPerFrame(40));
     09 }
}
```

As you can see in the preceding listing, the `MdtCharInterDemon` class is an explicit definition of the `MdtCharInter` class that displays the demon enemy character. Be sure to take a look at the other enemy classes provided in this chapter's completed project, included in the game engine's project folder. I'll also provide a complete set of level 2 classes that can be copied and pasted into your game project at the end of this section.

MdtItemPotionGreen: Class Review

The next class up for review is the level 2 extended class `MdtItemPotionGreen`, which extends the `MdtCharInter` level 1 base class. The `MdtItemPotion` green class is also very concise, so I'll list the entire class here with adjustments for Java and C# included.

Listing 23-30. MdtItemPotionGreen Class Review 1

```
//Java Version
public class MdtItemPotionGreen extends MdtItemPotion {

//C# Version
public class MdtItemPotionGreen : MdtItemPotion {

      //Java Version
      01 public MdtItemPotionGreen() {
      02    super(MmgHelper. GetBasicCachedBmp ("potion_green_lg.png"),
            MdtItemPotionType.GREEN, MdtPointsType.PTS_100);

      //C# Version
      01 public MdtItemPotionGreen()
      02     : base(MmgHelper. GetBasicCachedBmp ("potion_green_lg.png"),
             MdtItemPotionType.GREEN, MdtPointsType.PTS_100) {

      03 }

      //Java Version
      01 public MdtItemPotionGreen(MmgBmp Subj) {
      02     super(Subj, MdtItemPotionType.GREEN, MdtPointsType.PTS_100);

      //C# Version
      01 public MdtItemPotionGreen(MmgBmp Subj)
      02     : base(Subj, MdtItemPotionType.GREEN, MdtPointsType.PTS_100) {

      03 }
}
```

This level 2 class implementation is not unlike the one we've reviewed previously. Again, notice that the class doesn't define any new functionality; it simply specifies the resources necessary to display a green potion item on the screen that the player can interact with. The other two classes in this set that I didn't cover here are the `MdtItemPotionRed` and `MdtItemPotionYellow` classes. Be sure to include these classes in your project or wait till the end of this section where I'll provide instructions on how to get a copy of all of the level 2 classes. The next class we'll look into is an example of a level 2 class that extends the `MdtObjPush` level 1 base class.

MdtObjPushBarrel: Class Review

The `MdtObjPushBarrel` class is a level 2 extended class that extends the `MdtObjPush` level 1 base class. Again, this is an example of a concrete implementation of a base class. The entire class is short, so we can list it here with adjustments for the C# version included.

Listing 23-31. MdtObjPushBarrel Class Review 1

```
//Java Version
public class MdtObjPushBarrel extends MdtObjPush {

//C# Version
public class MdtObjPushBarrel : MdtObjPush {

    01 public MdtObjPushBarrel(ScreenGame Screen) {
    02     SetSubj(MmgHelper.GetBasicCachedBmp("barrel_lg.png"));
    03     SetMdtType(MdtObjType.OBJECT);
    04     SetMdtSubType(MdtObjSubType.OBJECT_BARREL);
    05     SetScreen(Screen);
    06     SetWidth(subj.GetWidth());
    07     SetHeight(subj.GetHeight());
    08     SetPushSpeed(ScreenGame.GetSpeedPerFrame(280));
    09
    10     MmgBmp src = MmgHelper.GetBasicCachedBmp("explosion_anim_
           spritesheet_lg.png");
    11     MmgSpriteSheet ssSrc = new MmgSpriteSheet(src, 32, 32);
    12     subjBreaks = new MmgSprite(ssSrc.GetFrames());
    13     subjBreaks.SetMsPerFrame(50);
    14 }
```

```
    //Java Version
    01 public MdtObjPushBarrel(MmgBmp Subj, MmgSprite SubjBreaks,
       ScreenGame Screen) {
    02      super(Subj, SubjBreaks, MdtObjType.OBJECT, MdtObjSubType.
            OBJECT_BARREL, Screen);

    //C# Version
    01 public MdtObjPushBarrel(MmgBmp Subj, MmgSprite SubjBreaks,
    ScreenGame Screen)
    02     : base(Subj, SubjBreaks, MdtObjType.OBJECT, MdtObjSubType.
    OBJECT_BARREL, Screen) {

03 }
}
```

Make sure to review the other level 2 extended classes that extend the `MdtObjPush` class before adding them to your project. Take a moment to look at the `MdtObjPushTableSmall` and `MdtObjPushTableLarge` classes in the chapter completed code. You'll need to implement these classes as well. There is one more level 2 class for us to review. In the next section, we'll tackle the user-controlled `MdtChatInterPlayer` class.

MdtCharInterPlayer: Class Review

The `MdtCharInterPlayer` class is a level 2 extended class that extends the `MdtCharInter` level 1 base class. The `MdtCharInterPlayer` class adds new functionality to the base class by supporting modifiers. In the DungeonTrap game, the potion items will trigger a modifier on the player character that picks up, collides with, the item. The `MdtCharInterPlayer` class does not have any static class members or enumerations to cover, so we'll start the review process in the class fields section.

MdtCharInterPlayer: Class Fields

The player character class adds functionality that supports three modifiers and class fields that power the ability to push an `MdtObjPush` object instance.

Listing 23-32. MdtCharInterPlayer Class Fields 1

```
//Full Health Mod
public boolean hasFullHealth = false;
public long modTimingFullHealth = 0;
public long modTimingFullHealthTotal = 3000;

//Invincibility Mod
public boolean hasInvincibility = false;
public long modTimingInv = 0;
public long modTimingInvTotal = 10000;

//Double Points Mod
public boolean hasDoublePoints = false;
public long modTimingDp = 0;
public long modTimingDpTotal = 15000;
```

The first block of class fields, listed in the preceding, are used to track three different types of player modifiers. Each modifier has a Boolean flag associated with it and two timing fields. One field is used to measure how long the given modifier has been active, while the second timing field holds the maximum time the given modifier can be active.

Listing 23-33. MdtCharInterPlayer Class Fields 2

```
//Push Start/Pushing Fields
public boolean isPushing = false;
public boolean isPushStart = false;
public long pushingStartMs;
public long pushingCurrentMs;
public long pushingLengthMs = 150;

//Mod Tracking Fields
public MdtPlayerModType mod = MdtPlayerModType.NONE;
public MdtPlayerModType prevMod = MdtPlayerModType.NONE;
```

The second block of class fields are used to track the character's push interaction along with fields that track the character's overall modifier state. The push functionality is described as having a push start phase, which is denoted by the `isPushStart` Boolean and `pushingStartMs` class fields.

The pushing phase of the functionality is described with the `isPushing` Boolean and `pushingCurrentMs` class fields. The `pushingLengthMs` class field is used to measure how much time it takes to go from the push start state to the pushing state. The last two class fields listed, `mod` and `prevMod`, are used to track the character's current modifier as well as its previous modifier. In the next section, we'll take a look at the class' pertinent methods.

MdtCharInterPlayer: Pertinent Method Outline

Let's take a look at the `MdtCharInterPlayer` class' methods listed in the following in two groups, main and support.

Listing 23-34. MdtCharInterPlayer Pertinent Method Outline 1

```
//Main Methods
public MdtCharInterPlayer(MmgSprite Subj, int FrameFrontS, int FrameFrontE,
int FrameBackS, int FrameBackE, int FrameLeftS, int FrameLeftE, int
FrameRightS, int FrameRightE, ScreenGame Screen, MdtPlayerType Player) { ... }

public MdtCharInterPlayer(MmgSprite Subj, int FrameFrontS, int FrameFrontE,
int FrameBackS, int FrameBackE, int FrameLeftS, int FrameLeftE, int
FrameRightS, int FrameRightE, ScreenGame Screen, Hashtable<String,
MdtWeapon> Weapons, String WeaponKey) { ... }

public void Bounce(MmgVector2 collPos, int halfWidth, int halfHeight, int
bounceDir, MdtPlayerType BounceBy) { ... }

public void ClearInvincibilityEffect(MmgSprite subj) { ... }

public boolean MmgUpdate(int updateTick, long currentTimeMs, long
msSinceLastFrame) { ... }

//Support Methods
public boolean GetHasFullHealth() { ... }
public void SetHasFullHealth(boolean b) { ... }
public long GetModTimingFullHealth() { ... }
public void SetModTimingFullHealth(long i) { ... }
public long GetModTimingFullHealthTotal() { ... }
public void SetModTimingFullHealthTotal(long i) { ... }
```

```
public boolean GetHasInvincibility() { ... }
public void SetHasInvincibility(boolean b) { ... }
public long GetModTimingInv() { ... }
public void SetModTimingInv(long i) { ... }
public long GetModTimingInvTotal() { ... }
public void SetModTimingInvTotal(long i) { ... }

public boolean GetHasDoublePoints() { ... }
public void SetHasDoublePoints(boolean b) { ... }
public long GetModTimingDp() { ... }
public void SetModTimingDp(long i) { ... }
public long GetModTimingDpTotal() { ... }
public void SetModTimingDpTotal(long i) { ... }

public long GetPushingStartMs() { ... }
public void SetPushingStartMs(long l) { ... }
public long GetPushingCurrentMs() { ... }
public void SetPushingCurrentMs(long l) { ... }
public long GetPushingLengthMs() { ... }
public void SetPushingLengthMs(long l) { ... }
public boolean GetIsPushing() { ... }
public void SetIsPushing(boolean b) { ... }
public MdtPlayerModType GetMod() { ... }
public void SetMod(MdtPlayerModType Mod) {  ... }
public MdtWeaponAttackType GetCurrentWeaponAttackType() { ... }
public MdtWeaponType GetCurrentWeaponType() { ... }
public MdtPlayerModType GetPrevMod() { ... }
public void SetPrevMod(MdtPlayerModType mod) { ... }
```

I'll also list the class definitions for both Java and C# in the following.

Listing 23-35. MdtCharInterPlayer Class Definitions 1

```
//Java Version
public class MdtCharInterPlayer extends MdtCharInter {

//C# Version
public class MdtCharInterPlayer : MdtCharInter {
```

That concludes our review of the class' pertinent method outline. Up next, we'll take a look at the class' support methods in greater detail.

MdtCharInterPlayer: Support Method Details

The `MdtCharInterPlayer` class has a number of simple get and set support methods listed in the following.

Listing 23-36. MdtCharInterPlayer Support Method Details 1

```
01 public MdtPlayerModType GetPrevMod() {
02     return prevMod;
03 }

01 public void SetPrevMod(MdtPlayerModType mod) {
02     prevMod = mod;
03     if(prevMod == MdtPlayerModType.DOUBLE_POINTS) {
04         hasDoublePoints = false;
05     } else if(prevMod == MdtPlayerModType.INVINCIBLE) {
06         hasInvincibility = false;
07     } else if(prevMod == MdtPlayerModType.FULL_HEALTH) {
08         hasFullHealth = false;
09     }
10 }

01 public boolean GetHasFullHealth() {
02     return hasFullHealth;
03 }

01 public void SetHasFullHealth(boolean b) {
02     hasFullHealth = b;
03 }

01 public long GetModTimingFullHealth() {
02     return modTimingFullHealth;
03 }
```

```
01 public void SetModTimingFullHealth(long i) {
02     modTimingFullHealth = i;
03 }

01 public long GetModTimingFullHealthTotal() {
02     return modTimingFullHealthTotal;
03 }

01 public void SetModTimingFullHealthTotal(long i) {
02     modTimingFullHealthTotal = i;
03 }

01 public boolean GetHasInvincibility() {
02     return hasInvincibility;
03 }

01 public void SetHasInvincibility(boolean b) {
02     hasInvincibility = b;
03 }

01 public boolean GetHasDoublePoints() {
02     return hasDoublePoints;
03 }

01 public void SetHasDoublePoints(boolean b) {
02     hasDoublePoints = b;
03 }

01 public long GetModTimingDp() {
02     return modTimingDp;
03 }

01 public void SetModTimingDp(long i) {
02     modTimingDp = i;
03 }

01 public long GetModTimingDpTotal() {
02     return modTimingDpTotal;
03 }
```

```
01 public void SetModTimingDpTotal(long i) {
02     modTimingDpTotal = i;
03 }

01 public long GetPushingStartMs() {
02     return pushingStartMs;
03 }

01 public void SetPushingStartMs(long l) {
02     pushingStartMs = l;
03 }

01 public long GetPushingCurrentMs() {
02     return pushingCurrentMs;
03 }

01 public void SetPushingCurrentMs(long l) {
02     pushingCurrentMs = l;
03 }

01 public long GetPushingLengthMs() {
02     return pushingLengthMs;
03 }

01 public void SetPushingLengthMs(long l) {
02     pushingLengthMs = l;
03 }

01 public boolean GetIsPushing() {
02     return isPushing;
03 }

01 public void SetIsPushing(boolean b) {
02     isPushing = b;
03 }

01 public MdtPlayerModType GetMod() {
02     return mod;
03 }
```

```
01 public void SetMod(MdtPlayerModType Mod) {
02     SetPrevMod(mod);
03     mod = Mod;
04 }

01 public long GetModTimingInv() {
02     return modTimingInv;
03 }

01 public void SetModTimingInv(long i) {
02     modTimingInv = i;
03 }

01 public long GetModTimingInvTotal() {
02     return modTimingInvTotal;
03 }

01 public void SetModTimingInvTotal(long i) {
02     modTimingInvTotal = i;
03 }

01 public MdtWeaponAttackType GetCurrentWeaponAttackType() {
02     return weaponCurrent.attackType;
03 }

01 public MdtWeaponType GetCurrentWeaponType() {
02     return weaponCurrent.weaponType;
03 }
```

Make sure you read and review the support methods listed in the preceding before you add them to your copy of the `MdtCharInterPlayer` class. Next up, we'll take a look at the class' main methods.

Listing 23-37. MdtCharInterPlayer Main Method Details 1

```
01 public MdtCharInterPlayer(MmgSprite Subj, int FrameFrontS, int
FrameFrontE, int FrameBackS, int FrameBackE, int FrameLeftS, int
FrameLeftE, int FrameRightS, int FrameRightE, ScreenGame Screen,
MdtPlayerType Player) {
```

```
02     super(Subj, FrameFrontS, FrameFrontE, FrameBackS, FrameBackE,
FrameLeftS, FrameLeftE, FrameRightS, FrameRightE, Screen, MdtObjType.
PLAYER, MdtObjSubType.PLAYER_1);
03     SetPlayerType(Player);
04
05     if(Player == MdtPlayerType.PLAYER_1) {
06         SetMdtSubType(MdtObjSubType.PLAYER_1);
07     } else {
08        SetMdtSubType(MdtObjSubType.PLAYER_2);
09     }
10
11     SetHealthMax(4);
12     SetHealthCurrent(4);
13 }
```

```
01 public MdtCharInterPlayer(MmgSprite Subj, int FrameFrontS, int
FrameFrontE, int FrameBackS, int FrameBackE, int FrameLeftS, int
FrameLeftE, int FrameRightS, int FrameRightE, ScreenGame Screen,
Hashtable<String, MdtWeapon> Weapons, String WeaponKey) {
02     super(Subj, FrameFrontS, FrameFrontE, FrameBackS, FrameBackE,
FrameLeftS, FrameLeftE, FrameRightS, FrameRightE, Screen, MdtObjType.
PLAYER, MdtObjSubType.PLAYER_1);
03     SetPlayerType(MdtPlayerType.PLAYER_1);
04     SetHealthMax(4);
05     SetHealthCurrent(4);
06     weaponCurrent.SetPlayer(GetPlayerType());
07 }
```

```
@Override
01 public void Bounce(MmgVector2 collPos, int halfWidth, int halfHeight,
int bounceDir, MdtPlayerType BounceBy) {
02     super.Bounce(collPos, halfWidth, halfHeight, bounceDir, BounceBy);
03     isPushStart = false;
04     isPushing = false;
05     pushingCurrentMs = 0;
06 }
```

The first constructor listed in the preceding takes a number of arguments that are used to call the super class, or base class if you're following along in C#. On lines 3–9, the type of character and object subtype are determined. The last two lines of code set the character's health to full value, four health points. The second constructor, also listed in the preceding, is similar to the first one except that it defaults fields to player1.

The next entry in the main methods we need to review is the `Bounce` method. Note that this method overrides the super class' implementation but also calls the super class method. This is an example of extending the super class method functionality. In this case, we turn off the push start and pushing states for the bouncing character.

Pushing an `MdtObjPush` instance is another new feature that is added to the `MdtCharInter` class by the `MdtCharInterPlayer` implementation. The following block of main methods has a few important lines of code we need to cover. Let's take a look.

Listing 23-38. MdtCharInterPlayer Main Method Details 2

```
01 public void ClearInvincibilityEffect(MmgSprite subj) {
02     MmgBmp[] bmps = subj.GetBmpArray();
03     int len = bmps.length;
04     for(int i = 0; i < len; i++) {
05         bmps[i].SetMmgColor(null);
06     }
07 }
```

```
@Override
01 public boolean MmgUpdate(int updateTick, long currentTimeMs, long
   msSinceLastFrame) {
02     lret = false;
03     if (isVisible == true) {
04         super.MmgUpdate(updateTick, currentTimeMs, msSinceLastFrame);
05         if(!isBroken) {
06             if(mod == MdtPlayerModType.INVINCIBLE) {
07                 modTimingInv += msSinceLastFrame;
08                 if(modTimingInv <= modTimingInvTotal) {
09                     int r = GetRand().nextInt(11);
10                     if(r % 3 == 0) {
11                         subj.GetCurrentFrame().SetMmgColor(MmgColor.
                           GetYellow());
```

```
12                  } else if(r % 3 == 1) {
13                      subj.GetCurrentFrame().SetMmgColor(MmgColor.
                        GetWhite());
14                  } else if(r % 3 == 2) {
15                      subj.GetCurrentFrame().SetMmgColor(MmgColor.
                        GetOrange());
16                  } else {
17                      subj.GetCurrentFrame().SetMmgColor(MmgColor.
                        GetRedOrange());
18                  }
19              } else {
20                  modTimingInv = 0;
21                  mod = MdtPlayerModType.NONE;
22                  hasInvincibility = false;
23                  screen.UpdateClearPlayerMod(playerType);
24                  ClearInvincibilityEffect(subj);
25              }
26          } else if(mod == MdtPlayerModType.FULL_HEALTH) {
27              modTimingFullHealth += msSinceLastFrame;
28              if(modTimingFullHealth <= modTimingFullHealthTotal) {
29                  healthCurrent = healthMax;
30              } else {
31                  modTimingFullHealth = 0;
32                  mod = MdtPlayerModType.NONE;
33                  hasFullHealth = false;
34                  screen.UpdateClearPlayerMod(playerType);
35              }
36          } else if(mod == MdtPlayerModType.DOUBLE_POINTS) {
37              modTimingDp += msSinceLastFrame;
38              if(modTimingDp > modTimingDpTotal) {
39                  modTimingDp = 0;
40                  mod = MdtPlayerModType.NONE;
41                  hasDoublePoints = false;
42                  screen.UpdateClearPlayerMod(playerType);
43              }
44          }
```

```
45
46              if(prevMod == MdtPlayerModType.INVINCIBLE && mod !=
                MdtPlayerModType.INVINCIBLE) {
47                  ClearInvincibilityEffect(subj);
48              }
49          }
50      }
51      return lret;
52  }
```

The first method listed in the preceding block of code is the `ClearInvincibilityEffect` method. To create an animation that shows the character is invincible, we've added code to change the color of the character's animation frames to a random color. This has the effect of creating a flashing pattern behind the character's image.

When the player's invincibility modifier runs out, we need to reset the character's animation frames so that the random colors are gone. The reset occurs on line 5 where the `mmgColor` field is set to null. The second method listed in the preceding is the `MmgUpdate` method that is responsible for updating the object on each game frame.

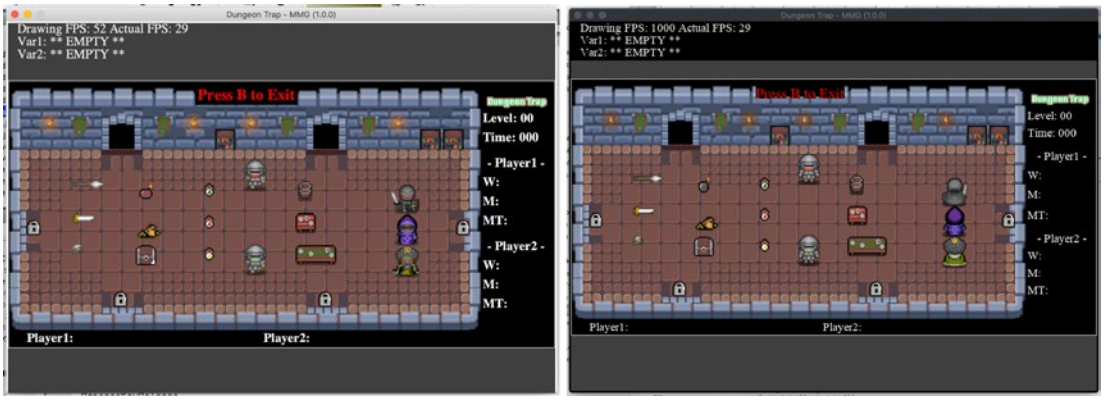
The method seems complicated, but if you step back and look at the overall structure of the class, you'll see that it simply runs the timing checks for the three different player modifiers. In each case, a specific action is taken; but the overall structure of measuring the elapsed time of the modifier and then turning it off after a period of time is shared between all three modifiers. On lines 46–48, there are a few special lines of code that prevent the invincibility effect from being cleared when switching to the same modifier.

This brings us to the conclusion of the `MdtCharInterPlayer` class. You don't have to spend an inordinate amount of time typing up code. There's a complete copy of all the level 2 classes found in the game engine project's "cfg" folder. Locate and open the "dungeon_trap_level2_classes" folder. Copy contents of the folder into your game project alongside your other project classes. Double-check that the `package/namespace` of the newly pasted files matches your project configuration.

We have one more thing to do before we can run the game and check out the results of all our hard work. We need to update our copy of the `ScreenGame` class to include placeholders for missing class methods. I'll do you one better than that. I've made a special `ScreenGame` file that includes almost all the final game code for the class' fields and resource loading method. It's also configured to demo all the game objects we've built in this chapter and the previous one.

In the same folder where you found the "dungeon_trap_level2_classes" folder, you will find a "dungeon_trap_chapter23_demo_screen" folder. Copy the contents of this folder and paste the classes into your game project, overwriting any existing files. Be sure to double-check the `package/namespace` of the newly pasted files as they most likely will not have the proper values set.

Take care of any errors. Reference the completed chapter code if you need to. Now compile your project and run it. When the game's menu screen appears, click the first menu option, and you'll be taken to a game screen that demos all the game objects you've created. Take a look at the following screenshot.



this figure will be printed in b/w

This chapter takes care of a number of game specifications from the list we looked at long ago in Chapter 15. There are now game classes that define items, enemies, weapons, and objects. That knocks off a few specifications from the list. We now have the foundation to build the interactions between the game objects that will express the actual game. And that's exactly what we'll do in Chapter 24!