CHAPTER 21

DungeonTrap Main Menu Screen

In this chapter, we'll move the project even further along by adding a custom main menu screen to the game. This chapter is similar to Chapter 17 except it's specifically designed for the DungeonTrap game. Let's start things off by taking a look at the main menu screen and what we need to do to get it set up.

The following process should work fine with both the Java and C# versions of the project. Create a new class in the same location as the project's existing classes and name it "ScreenMainMenu." We want this class to extend the MmgCore API's `ScreenMainMenu` class. Keep in mind that we'll have access to that functionality as we define how our local `ScreenMainMenu` class works. Because the screen's default implementation includes the following features – title, subtitle, single player, game start, game exit, footer, and version number – we won't have to worry too much about them.

Let's take a moment to look at our game's general specifications listed in Chapter 15 and see what we're missing. One thing that stands out to me is that the game is defined as supporting both one- and two-player modes. However, the default main menu screen only has an option for starting a single-player game. We'll have to adjust things a bit. I'll start by listing the class headers in the following. If you are unsure about the class' `using/import` statements, take a moment to view this chapter's completed project, included in the game engine project.

V. G. Brusca, *Introduction to Video Game Engine Development*, https://doi.org/10.1007/978-1-4842-7039-4_21

Listing 21-1. ScreenMainMenu Class Code 1

```
//Java Method Signature
public class ScreenMainMenu extends net.middlemind.MmgGameApiJava.MmgCore.
ScreenMainMenu {

//C# Method Signature
public class ScreenTestMmgMainMenu : net.middlemind.MmgGameApiCs.MmgCore.
ScreenMainMenu {

//Class Outline
private MmgBmp menuStartGame1P;
private MmgBmp menuStartGame2P;
private boolean lret;

public ScreenMainMenu(GameStates State, GamePanel Owner) { ... }

@Override
public void LoadResources() { ... }

@Override
public boolean ProcessAClick(int src) { ... }

@Override
public boolean ProcessDpadRelease(int dir) { ... }

@Override
public void DrawScreen() { ... }

@Override
public void UnloadResources() { ... }

@Override
public GameStates GetGameState() { ... }

@Override
public boolean GetIsDirty() { ... }

@Override
public void SetIsDirty(boolean b) { ... }

public void MakeDirty() { ... }
```

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

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

The methods outlined in the preceding should be familiar to you from our review of the MmgCore API's game screen classes. We're going to be methodically building up this class over the next few pages. Let's start with the class' constructor. Again, I won't be addressing the slight differences between the Java and C# versions. We'll stick to working with the Java code for the most part. If you run into trouble following along in either language, then please check the example project in the MmgTestSpace library or the completed chapter projects included with the game engine project.

Listing 21-2. ScreenMainMenu Class Code 2

```
01 public ScreenMainMenu(GameStates State, GamePanel Owner) {
02     super(State, Owner);
03     isDirty = false;
04     pause = false;
05     ready = false;
06     state = State;
07     owner = Owner;
08 }
```

The constructor, much like in other screen classes we've reviewed, calls the super class constructor and prepares all the class fields as shown in the preceding. I think we should get the class' support methods out of the way first. I'll list the methods in the following; please add them to your `ScreenMainMenu` class along with the constructor outlined in the preceding.

Listing 21-3. ScreenMainMenu Class Code 3

```
@Override
01 public GameStates GetGameState() {
02     return state;
03 }
```

```
@Override
01 public boolean GetIsDirty() {
02     return isDirty;
03 }

@Override
01 public void SetIsDirty(boolean b) {
02     isDirty = b;
03 }

01 public void MakeDirty() {
02     isDirty = true;
03 }
```

Some of these methods are already defined by the super class, or base class in C#. We didn't need to override them and redefine them in this class, but I wanted to make sure that the code was local in case I needed to make customizations. In the end, it's only a few lines of code, so let's not worry too much about it. Next up, let's take a look at the class' input methods. Notice that in this case we're overriding the super class and not using super class functionality. We're letting the local class take full control.

Listing 21-4. ScreenMainMenu Class Code 4

```
@Override
01 public boolean ProcessAClick(int src) {
02     int idx = GetMenuIdx();
03     MmgMenuItem mmi;
04     mmi = (MmgMenuItem) menu.GetContainer().get(idx);
05
06     if (mmi != null) {
07         ProcessMenuItemSel(mmi);
08         return true;
09     }
10
11     return false;
12 }
```

```
@Override
01 public boolean ProcessDpadRelease(int dir) {
02     if (dir == GameSettings.UP_KEYBOARD || dir == GameSettings.UP_
       GAMEPAD_1) {
03         MoveMenuSelUp();
04     } else if (dir == GameSettings.DOWN_KEYBOARD || dir == GameSettings.
       DOWN_GAMEPAD_1) {
05         MoveMenuSelDown();
06     }
07
08     return true;
09 }
```

Review the input methods listed in the preceding and add them to your ScreenMainMenu class. This adds input handling to the menu screen. The next set of class methods for us to implement are the LoadResources and UnloadResources methods. If you recall from the ScreenMainMenu class' review, the LoadResources method is responsible for loading and configuring the screen's resources. Let's take a look at a snippet of code from this method.

Listing 21-5. ScreenMainMenu Class Code 5

```
@Override
01 public void LoadResources() {
02     pause = true;
03     SetHeight(MmgScreenData.GetGameHeight());
04     SetWidth(MmgScreenData.GetGameWidth());
05     SetPosition(MmgScreenData.GetPosition());
06
07     classConfig = MmgHelper.ReadClassConfigFile(GameSettings.CLASS_
       CONFIG_DIR + GameSettings.NAME + "/screen_main_menu.txt");
08
09     MmgBmp tB = null;
10     MmgPen p;
11     String key = "";
12     String imgId = "";
13     String sndId = "";
```

```
14      MmgBmp lval = null;
15      MmgSound sval = null;
16      String file = "";
17
18      p = new MmgPen();
19      p.SetCacheOn(false);
20      handleMenuEvent = new HandleMainMenuEvent(this, owner);
21
22      key = "soundMenuSelect";
23      if(classConfig.containsKey(key)) {
24          file = classConfig.get(key).str;
25      } else {
26          file = "jump1.wav";
27      }
28
29      sndId = file;
30      sval = MmgHelper.GetBasicCachedSound(sndId);
31      menuSound = sval;
32
33      tB =  MmgHelper.CreateFilledBmp(w, h, MmgColor.GetBlack());
34      if (tB != null) {
35          SetCenteredBackground(tB);
36      }
37
38      key = "bmpGameTitle";
39      if(classConfig.containsKey(key)) {
40          file = classConfig.get(key).str;
41      } else {
42          file = "game_title.png";
43      }
44
45      imgId = file;
46      lval = MmgHelper.GetBasicCachedBmp(imgId);
47      menuTitle = lval;
48      if (menuTitle != null) {
49          MmgHelper.CenterHor(menuTitle);
```

```
50          menuTitle.GetPosition().SetY(GetPosition().GetY() + MmgHelper.
            ScaleValue(40));
51          menuTitle = MmgHelper.ContainsKeyMmgBmpScaleAndPosition
            ("menuTitle", menuTitle, classConfig, menuTitle.GetPosition());
52          AddObj(menuTitle);
53      }
```

This is quite a long method, so take your take your time with it. I recommend copying the full version of this method from the Chapter 21 project included with the game engine project. The main take-away from the preceding listing is on lines 45–53. The code block shows how resources are loaded and processed. The same steps are applied to each image resource on this screen. It's redundant and we've seen similar code during the MmgCore API review, so I won't go over it again here.

Listing 21-6. ScreenMainMenu Class Code 6

```
@Override
01 public void UnloadResources() {
02      isDirty = false;
03      pause = true;
04
05      SetIsVisible(false);
06      SetBackground(null);
07      SetMenu(null);
08      ClearObjs();
09
10      menuStartGame = null;
11      menuStartGame1P = null;
12      menuStartGame2P = null;
13      menuExitGame = null;
14      menuTitle = null;
15      menuFooterUrl = null;
16      menuCursor = null;
17      menuSound = null;
18
19      handleMenuEvent = null;
20      classConfig = null;
```

```
21
22      super.UnloadResources();
23
24      menu = null;
25      ready = false;
26  }
```

The UnloadResources method is used to reverse what the LoadResources method does and clear any class resources. Notice that there is one line in the method that stands out a bit. Remember the class you're working on extends a super class, so we must be sure to call the super class' UnloadResources method to ensure all resources have been released. There is a specific call to the super class method on line 22.

We're just about finished with this class, but we still have a bit of work to do. The next method we'll work on is the DrawScreen method. We've reviewed this method before as well, but in case you're a little fuzzy on its details, think of this method as a runtime configuration method. It will redraw the screen in response to an MmgUpdate call if the class is marked as dirty. Let's take a look at some code!

Listing 21-7. ScreenMainMenu Class Code 7

```
@Override
01 public void DrawScreen() {
02      pause = true;
03      menu = new MmgMenuContainer();
04      menu.SetMmgColor(null);
05      isDirty = false;
06
07      MmgMenuItem mItm = null;
08
09      if (menuStartGame1P != null) {
10          mItm = MmgHelper.GetBasicMenuItem(handleMenuEvent, "Main Menu
            Start Game 1P", HandleMainMenuEvent.MAIN_MENU_EVENT_START_GAME_1P,
            HandleMainMenuEvent.MAIN_MENU_EVENT_TYPE, menuStartGame1P);
11          mItm.SetSound(menuSound);
12          menu.Add(mItm);
13      }
14
```

```
15      if (menuStartGame2P != null) {
16          mItm = MmgHelper.GetBasicMenuItem(handleMenuEvent, "Main Menu
            Start Game 2P", HandleMainMenuEvent.MAIN_MENU_EVENT_START_GAME_2P,
            HandleMainMenuEvent.MAIN_MENU_EVENT_TYPE, menuStartGame2P);
17          mItm.SetSound(menuSound);
18          menu.Add(mItm);
19      }
20
21      if (menuExitGame != null) {
22          mItm = MmgHelper.GetBasicMenuItem(handleMenuEvent, "Main Menu
            Exit Game", HandleMainMenuEvent.MAIN_MENU_EVENT_EXIT_GAME,
            HandleMainMenuEvent.MAIN_MENU_EVENT_TYPE, menuExitGame);
23          mItm.SetSound(menuSound);
24          menu.Add(mItm);
25      }
26
27      SetMenuStart(0);
28      SetMenuStop(menu.GetCount() - 1);
29
30      SetMenu(menu);
31      SetMenuOn(true);
32      pause = false;
33  }
```

Notice that the `DrawScreen` method is responsible for setting up the menu items after the first call to the `MmgUpdate` method when the class is marked as dirty. Add this method to the `ScreenMainMenu` class, and we'll move on to tackle the drawing routine methods `MmgDraw` and `MmgUpdate`. Let's take a look at the `MmgDraw` method next.

Listing 21-8. ScreenMainMenu Class Code 8

```
01 public void MmgDraw(MmgPen p) {
02      if (pause == false && GetIsVisible() == true) {
03          super.MmgDraw(p);
04      }
05  }
```

Add the MmgDraw method listed in the preceding to your ScreenMainMenu class and save your work. Please remember to save early and save often. The last method we have to work on is the MmgUpdate method listed next.

Listing 21-9. ScreenMainMenu Class Code 9

```
01 public boolean MmgUpdate(int updateTick, long currentTimeMs, long
   msSinceLastFrame) {
02     lret = false;
03
04     if (pause == false && isVisible == true) {
05         if (super.MmgUpdate(updateTick, currentTimeMs, msSinceLastFrame)
           == true) {
06             lret = true;
07         }
08
09         if (isDirty == true) {
10             lret = true;
11             DrawScreen();
12         }
13
14     }
15
16     return lret;
17 }
```

We're almost ready to compile and test our project, but we have a few more steps to complete before the new ScreenMainMenu game screen is plugged into the game. Open up the GamePanel class and add the following line of code to the bottom of the list of class fields:

```
public ScreenMainMenu screenMainMenu;
```

You'll also need to add the following code to the GamePanel class' constructor. This will initialize the main menu screen along with the other game screens:

```
screenMainMenu = new ScreenMainMenu(GameStates.MAIN_MENU, this);
```

To connect the game screen into the GamePanel's known states, we need to add the following code to the cleanup section of the SwitchGameState method. Add in the following code if it doesn't already exist.

Listing 21-10. GamePanel Class Code 1

```
} else if (prevGameState == GameStates.MAIN_MENU) {
    MmgHelper.wr("Hiding MAIN_MENU screen.");
    screenMainMenu.Pause();
    screenMainMenu.SetIsVisible(false);
    screenMainMenu.UnloadResources();

}
```

We're almost there. We still need to add the game state preparation code that's responsible for getting the main menu screen ready to display. Add in the following code if it doesn't already exist.

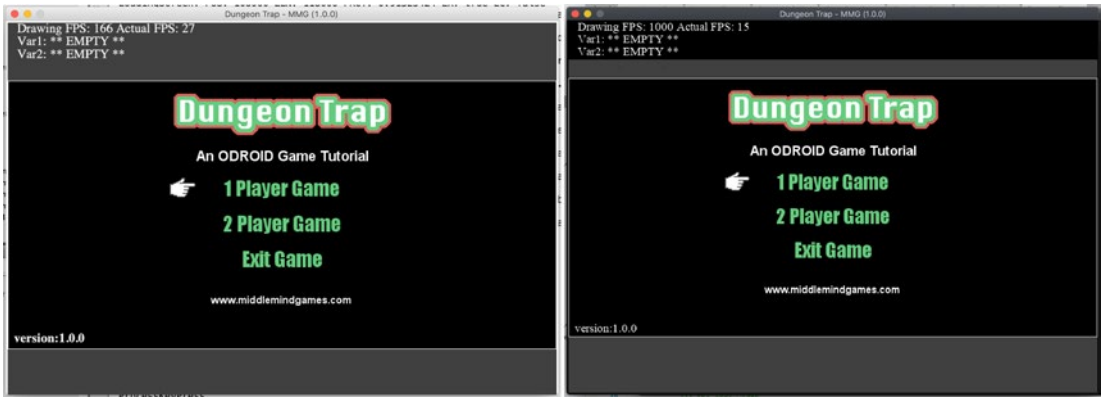
Listing 21-11. GamePanel Class Code 2

```
} else if (gameState == GameStates.MAIN_MENU) {
    MmgHelper.wr("Showing MAIN_MENU screen.");
    screenMainMenu.LoadResources();
    screenMainMenu.UnPause();
    screenMainMenu.SetIsVisible(true);
    currentScreen = screenMainMenu;

}
```

Now the new main menu screen is fully wired into the GamePanel class and subsequently the game. Once you're done adding in the SwitchGameState method updates, make sure your project compiles correctly. Address any errors if you encounter them, and if you run into real trouble, take a look at the example project for this chapter and double-check your code. Run your project using the same steps outlined in the previous chapter.

If you're following along in C#, then use the terminal to run your newly compiled project. If you're working with the Java version of the game, you can run the project directly in the IDE. Notice that the main menu is visible just like in the previous chapter, but in this case, it's now customized for DungeonTrap.



That brings us to the conclusion of this chapter. We've addressed a number of points from the DungeonTrap game's specification list. In the next few chapters, we'll review the game-specific classes in preparation for implementing the main game screen.