Dragon's Claw – Web Tutorial Series
Chapter 1
Setup

This tutorial series is about creating a sci-fi themed text based strategic role playing game. The tutorials will make more sense if you read them in order as each tutorial builds on the previous tutorials. You can find the list of tutorials on my blog: Dragon's Claw. The source code for the tutorial series is available as a Git repository. I will be using Visual Studio 2019 Community for the series. The code should compile on the 2015 and 2017 versions as well.

I want to mention though that the series is released as Creative Commons 3.0 Attribution. It means that you are free to use any of the code or graphics in your own game, even for commercial use, with attribution. Just give credit to Cynthia McMahon and add a link to my site, https://mygameprogrammingadventures.blogspot,com. Screenshots of your project and/or a video of game play would be appreciated.

I also want to mention that I assume you have a basic understanding of C#, SQL, HTML, CSS, Javascript and jQuery. If you don't I recommend that you learn basic of each. Enough to be familiar with the syntax and how to read code.

This tutorial will cover setting up the project and adding in some basic functionality. First I will describe the way the game works. The game takes place in a galaxy that is filled with empires. Empires are made up of sectors. Sectors are made up of planets. Sectors have a civilian and a military population. Players control a sector in an empire. Players have a race and a class.

Start Visual Studio 2019. When the dialog comes up click the Create a new project. On this page enter MVC in the search box. From the list that comes up choose ASP.NET Web Application (.NET Framework) then click Next. On this page enter DragonsClaw for the project name, select the Place solution and project in same directory check box. For Framework select .NET 4.6.2. Click the Create button.

On this page select the MVC from the list on the left. Check the Web API check box. Leave the Configure for HTTPS check box selected. Click the Change link under Authentication. Choose the Individual user accounts option on the dialog that pops up and click the OK button. Click the Create button to continue.

Once the project has finished building open SQL Server Management Studio. When prompted to connect to SQL Server enter (LocalDb)\MSSQLLocalDB for the server name and choose Windows Authentication. Now create a new database called DragonsClaw. Do that by right clicking the Databases node in the Object Explorer, selecting New Database. In the window that pops up enter DragonsClaw for Database name and hitting OK.

Back in Visual Studio open Web.config. Replace the default connection with the following.

```
    <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;Initial
Catalog=DragonsClaw;Integrated Security=True"
      providerName="System.Data.SqlClient" />
```

Now press F5 to run the project. You may receive a warning message box about SSL. Check the check box and press OK. You might also get a Security Warning dialog about installing a certificate from a certificate authority claiming to be localhost. Click the Yes button to install the certificate. Click the Register link and register a new user account. You may get a Null reference exception in Visual Studio. If you do just click the Continue button to keep going. Once you've registered close the browser.

This will create some tables for users in the DragonsClaw database for users. We won't be doing much with these tables. I want to create a few other tables. The first is the table for players. Go to SQL Server Management Studio. Create a new query window by clicking New Query or pressing CTRL+N. Paste the following code in the window and press F5.

```
USE [DragonsClaw]
GO

/****** Object:  Table [dbo].[Players]    Script Date: 2020-08-29 9:17:54 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Players](
       [PlayerId] [int] IDENTITY(1,1) NOT NULL,
       [UserGuid] [nvarchar](50) NOT NULL,
       [UserEmail] [nvarchar](500) NOT NULL,
       [PlayerName] [nvarchar](50) NOT NULL,
       [SectorName] [nvarchar](50) NOT NULL,
       [Credits] [int] NOT NULL,
       [Planets] [int] NOT NULL,
       [PsionicEnergy] [int] NOT NULL,
       [Population] [int] NOT NULL,
       [Employed] [int] NOT NULL,
       [Citizens] [int] NOT NULL,
       [Cadets] [int] NOT NULL,
       [OffensiveTroops] [int] NOT NULL,
       [DefensiveTroops] [int] NOT NULL,
       [SpecialtyTroops] [int] NOT NULL,
       [Mercenaries] [int] NOT NULL,
       [Spies] [int] NOT NULL,
       [ClassId] [int] NOT NULL,
       [RaceId] [int] NOT NULL,
       [Psionists] [int] NOT NULL,
       [Gender] [int] NOT NULL,
       [Food] [int] NOT NULL,
       [PsionicCrystals] [int] NOT NULL,
       [Fighters] [int] NOT NULL,
       [Bombers] [int] NOT NULL,
       [Cruisers] [int] NOT NULL,
       [Destroyers] [int] NOT NULL,
       [Dreadnaughts] [int] NOT NULL,
       [Raiders] [int] NOT NULL,
       [Dilithium] [int] NOT NULL,
       [Ore] [int] NOT NULL,
```

```
	[Admirals] [int] NOT NULL,
	[Observer] [bit] NULL,
	[NetWorth] [int] NULL,
	[NetWorthPerPlanet] [int] NULL,
	[Stealth] [int] NULL,
	[TeraBytes] [int] NULL,
	[Happiness] [int] NULL,
 CONSTRAINT [PK_Players] PRIMARY KEY CLUSTERED
(
	[PlayerId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Gender]  DEFAULT ((1)) FOR [Gender]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Food]  DEFAULT ((0)) FOR [Food]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_PsionicCrystals]  DEFAULT ((0)) FOR
[PsionicCrystals]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Fighters]  DEFAULT ((0)) FOR [Fighters]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Bombers]  DEFAULT ((0)) FOR [Bombers]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Cruisers]  DEFAULT ((0)) FOR [Cruisers]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Destroyers]  DEFAULT ((0)) FOR
[Destroyers]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Dreadnaughts]  DEFAULT ((0)) FOR
[Dreadnaughts]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Raiders]  DEFAULT ((0)) FOR [Raiders]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Dilithium]  DEFAULT ((1000)) FOR
[Dilithium]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Ore]  DEFAULT ((10000)) FOR [Ore]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Admirals]  DEFAULT ((5)) FOR [Admirals]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_NetWorth]  DEFAULT ((0)) FOR [NetWorth]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_NetWorthPerPlanet]  DEFAULT ((0)) FOR
[NetWorthPerPlanet]
```

```
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Stealth]  DEFAULT ((100)) FOR [Stealth]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_TeraBytes]  DEFAULT ((0)) FOR
[TeraBytes]
GO

ALTER TABLE [dbo].[Players] ADD  CONSTRAINT [DF_Players_Happiness]  DEFAULT ((100)) FOR
[Happiness]
GO
```

There are a lot of columns in the table. I will explain them in the following table.

COLUMN	DESCRIPTION
PlayerId	The identity column for the table.
UserGuid	This is a unique identifier that identifies the user.
UserEmail	This is the user's email address.
PlayerName	This is the player's name in the game.
SectorName	This is the name of the player's sector in the game.
Credits	This is the amount of credits, or money, the player has.
Planets	This is the number of planets that the player has. Planets affect the population of the player's sector.
PsionicEnergy	Psionic energy is the amount of energy the player has for casting psionics, or spells.
Population	This is the total population of the players sector. It includes military and civilian population.
Employed	This is the number of citizens that are employed.
Citizens	This is the number of citizens in the sector.
Cadets	This is the number of cadets in the sector. Cadets are used for training troops and are drafted from the civilian population.
OffensiveTroops	This is the number of ground troops that specialize in offensive actions.
DefensiveTroops	This is the number of ground troops that specialize in defensive actions.
SpecialtyTroops	This is the number of troops that are specific to the player's race.
Mercenaries	This is the number of mercenaries that are available for combat. It is based on the amount of credits available.
Spies	This is the number of spies that are available for espionage.
RaceId	This is the identifier for the player's selected race.
ClassId	This is the identifier for the player's selected class.
Psionists	This is the number of psionists available for casting psionics, or spells.

Gender	This is the player's gender. 0 represents male, 1 represents female and 2 represents non-binary.
Food	This is the amount of food that the player has.
PsionicCrystals	This is the amount of psionic crystals that are available. Psionics require an amount of crystals to be cast.
Fighters	This is the number of space fighters that are available for combat.
Bombers	This is the number of bombers that are available for combat. They are offensive troops only.
Cruisers	This is the number of cruisers that are available for combat.
Destroyers	This is the number of destroyers that are available for combat.
Dreadnaughts	This is the number of dreadnaughts that are available for combat.
Raiders	This is the number of raiders that are available for raiding and defending against raids.
Dilithium	This is a resource that is required for building space fairing ships.
Ore	This is a resource that is required for building space fairing ships.
Admirals	This is the number of admirals that are available for attacks. The maximum number of admirals is 5.
Observer	This is if the player is an observer. Observers can not be attacked, raided, spied upon or have psionics cast on them. They also cannot attack, raid, spy or cast psionics on other players. It allows players to play an era and learn the game and be socialable.
Networth	This is the net worth of the player.
NetworthPerPlanet	This is the net worth per planet of the player.
Stealth	This is the player's stealth. Stealth is required for espionage and raiding.
Terabytes	Terabytes is the amount of data the player has to allocate towards research.
Happiness	Happiness is how content the population is and affects production.

I'm going to add the empire table next. In SQL Server Management Studio create a new query by pressing CTRL+N. Paste the following code into the window then press F5 to execute it.

```
USE [DragonsClaw]
GO

/****** Object:  Table [dbo].[Empires]    Script Date: 2020-08-29 11:06:37 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Empires](
	[EmpireId] [int] IDENTITY(1,1) NOT NULL,
```

```
	[EmpireName] [nvarchar](50) NOT NULL,
	[EmpireDescription] [nvarchar](500) NOT NULL,
	[HostId] [int] NOT NULL,
	[ViceHost1Id] [int] NOT NULL,
	[ViceHost2Id] [int] NOT NULL,
	[UnimatrixX] [int] NOT NULL,
	[UnimatrixY] [int] NOT NULL,
	[Decree] [nvarchar](1000) NULL,
 CONSTRAINT [PK_Empires] PRIMARY KEY CLUSTERED
(
	[EmpireId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

COLUMN NAME	DESCRIPTION
EmpireId	This is the identifier for the empire
EmpireName	The name of the empire.
EmpireDescription	The description of the empire.
HostId	The player identifier of the host. Hosts can make changes to the empire and appoint vice hosts.
ViceHost1Id	The player identifier of the first vice host. Vice hosts can make changes to the empire.
ViceHost2Id	The player identifier of the second vice host.
UnimatrixX	The X coordinate of the unimatrix of the empire.
UnimatrixY	The Y coordinate of the unimatrix of the empire.
Decree	The decree of the empire that is displayed on the empire window.

I created a relation table that links players and empires. In SQL Server Management Studio open a new query window by pressing CTRL+N. Paste the following code then press F5 to execute the query.

```
USE [DragonsClaw]
GO

/****** Object:  Table [dbo].[EmpirePlayer]    Script Date: 2020-08-29 11:32:36 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[EmpirePlayer](
	[EmpireId] [int] NOT NULL,
	[PlayerId] [int] NOT NULL
) ON [PRIMARY]
GO
```

The table has two columns. The first it the identifier of the empire and the second is the identifier of the player.

The two other tables that I want to add are the table for race and a table for classes. Open a new query window and paste the following code then press F5 to execute it.

```
USE [DragonsClaw]
GO

/****** Object:  Table [dbo].[Classes]    Script Date: 2020-08-29 11:41:34 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Classes](
	[ClassId] [int] IDENTITY(1,1) NOT NULL,
	[ClassName] [nvarchar](50) NOT NULL,
	[ClassDescription] [nvarchar](500) NOT NULL,
	[CreditModifier] [int] NOT NULL,
	[OffenseModifier] [int] NOT NULL,
	[DefenseModifier] [int] NOT NULL,
	[SpyModifier] [int] NOT NULL,
	[ScienceModifier] [int] NOT NULL,
	[PsionicMosifier] [int] NULL,
 CONSTRAINT [PK_Classes] PRIMARY KEY CLUSTERED
(
	[ClassId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

/****** Object:  Table [dbo].[Races]    Script Date: 2020-08-29 11:41:59 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Races](
	[RaceId] [int] IDENTITY(1,1) NOT NULL,
	[RaceName] [nvarchar](50) NOT NULL,
	[RaceDescription] [nvarchar](500) NOT NULL,
	[CreditModifier] [int] NOT NULL,
	[OffensiveModifier] [int] NOT NULL,
	[DefensiveModifier] [int] NOT NULL,
	[SpyModifier] [int] NOT NULL,
	[PsionicModifier] [int] NOT NULL,
	[ScienceModifier] [int] NOT NULL,
 CONSTRAINT [PK_Races] PRIMARY KEY CLUSTERED
(
	[RaceId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

The tables have an identifier column, name column and description column. There are also modifier columns for credits, offense, defense, spying, psionics and science. They affect the production or

efficiency of the selected item.

Time to insert a few races and classes into the database to work with. I did that by opening a new query window and running the following SQL statements.

```
USE [DragonsClaw]
GO

INSERT INTO [dbo].[Races] (RaceName, RaceDescription, CreditModifier, OffensiveModifier,
DefensiveModifier, SpyModifier, PsionicModifier, ScienceModifier)
       VALUES ('Human', 'Humans are the most common and diverse race', 0, 5, 5, 0, 0, 0),
                ('Vulcan', 'Highly logical and scientific.', 0, 0, 0, 0, 0, 10),
                ('Ferengi', 'Highly converned about monetary issues.', 10, 0, 0, 0, 0, 0),
                ('Wookie', 'Strong, hairy humanoids. Excellent warriors', 0, 10, 0, 0, 0, 0)
GO

INSERT INTO [dbo].[Classes] (ClassName, ClassDescription, CreditModifier, OffenseModifier,
DefenseModifier, SpyModifier, PsionicMosifier, ScienceModifier)
       VALUES ('Admiral', 'Uncanny tacticians gaining bonus to attack and defense.', 0, 5, 5,
0, 0, 0),
                ('Terraformer', 'Excels at infrastructure development and environmal
transformation.', 0, 0, 0, 0, 0, 10),
                ('Agent', 'Master of espionage.', 0, 0, 0, 10, 0, 0),
                ('Smuggler', 'Raiders extrordanaire', 10, 0, 0, 0, 0, 0)
GO
```

Just some simple inserts that insert some development data. I created four races: Human, Vulcan, Ferengi and Wookie. Humans are the base race and are fairly good at military affairs. Vulcans are from the Star Trek universe and are good at science. Ferengi are also from the Star Trek universe and are concerned about financial affairs. Wookies are from the Star Wars universe and are excellent warriors. There are four classes: Admiral, Terraformer, Agent and Smuggler. Admirals are tacticians and are good at war. Terraformers gain bonus to developing planets and science. Smugglers are exceptional at raiding and financial affairs.

That is it for database work for now. I will now turn my attention to code. I use LESS for my CSS files. I like being able to declare reusable variables and functions rather than using hard coded values. You can set up Visual Studio to compile LESS files for you by using a plugin by Mads Kristensen. You can download it from the following link:

https://marketplace.visualstudio.com/items?itemName=MadsKristensen.LESSCompiler

Download and run the installer. Just click the Install button to install the package. If you have blocking tasks click the End Task button to close them. Make sure to save your work first. Close the dialog.

Reopen Visual Studio and load the project. Right click the Content folder, select Add and the LESS Style Sheet. Name this new style sheet dragons-claw.less. Update the style sheet with the following code.

```
/* General CSS selectors */
/* Primary color - 0B04BE */
@background: #090397;
@highlight1: #3831FB;
@highlight2: #150CF1;
```

```
@highlightTransparent: rgba(56,49,251,0.25);
@accent1: #FFEC13;
@accent2: #FF9D13;
@accent3: #FFEB00;
@grey1: #616161;
@grey2: #686868;
@grey3: #c1c1c1;
@fontSize: 18px;
@largeFontSize: 24px;

* {
    margin: 0;
    padding: 0;
}

body {
    background-color: @background;
    color: white;
    font-family: 'postamtregular';
    font-size: @fontSize;
    width: 100%;
    margin: 0;
    overflow-y: scroll;
}

.highlight1 {
    color: @accent2;
}

.highlight2 {
    color: @accent1;
}
```

What this does is set up some variables that will be used through out the LESS file. They are mostly colors with some font sizes mixed in. It then resets the padding and margins of all elements. For the body it sets the background color to a shade of blue, the foreground color to white. I use a font that I downloaded from DaFont that is 100% free. You can find it at this link:

https://www.dafont.com/postamt.font

Download it and extract it to a folder. Now we will generate web fonts for this font. Browse to Font Squirrel Web Font Generator at this link:

https://www.fontsquirrel.com/tools/webfont-generator

Click the Upload button and browse to the postamt.ttf that you extracted. Click the Yes check box because the font is free then click the DOWNLOAD YOUR KIT button to create and download the package. Once you've downloaded the package and extracted its contents go back to Visual Studio. Right click the Content folder, select Add and then Existing item. Navigate to the folder that you extracted the fonts to and add the .woff and .woff2 files to the project. Go back to the dragons-claw.less file and add the following CSS.

```
@font-face {
    font-family: 'postamtregular';
```

```
    src: url('postamt-webfont.woff2') format('woff2'), url('postamt-webfont.woff')
format('woff');
    font-weight: normal;
    font-style: normal;
}
```

Now I'm going to add in the layout for the game, which is separate from the layout of the site. Expand the Views folder in the Solution Explorer. Right click the Shared folder, select Add and then View. In the dialog that pops up make sure to unselect the Use a layout page check box and select the Create partial view check box. Select Empty (without model) value for the Template drop down and name the view _GameLayout. Place the contents of this file in the view.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dragon's Claw</title>
    <script src="~/Scripts/jquery-3.4.1.min.js"></script>
    @if (Html.IsDebug())
    {
        <link rel="stylesheet/less" href="~/Content/dragons-claw.less" />
        <script src="//cdn.jsdelivr.net/npm/less"></script>
        <script src="~/Scripts/GameScript.js"></script>
    }
    else
    {
        <link rel="stylesheet" href="~/Content/dragons-claw.min.css" />
        <script src="~/Scripts/GameScript.es5.min.js"></script>
    }
</head>
<body>
    <div class="details-content">
        @RenderBody()
    </div>
    <div id="account-tab">
        <a href="#" class="account" onclick="toggleAccountTab();">Account</a>
        <a href="#" onclick="toggleAccountTab();" class="account align-right"
style="float:right;">V</a>
        <div id="account-details">
            <form action="/Account/LogOff" method="post">
                <input id="logoff" type="submit" value="Log Out" />
            </form>
        </div>
    </div>
</body>
</html>
```

This layout uses an extension method, that I will add next, IsDebug that is used to check if we are in debug mode or release mode. Why is this important? I use different files in release mode than in debug move. So there is an if statement that outputs different HTML in debug mode or release mode. In debug mode I use the LESS file and a JavaScript that I haven't added yet. In release mode I used the compiled version of the LESS file, minimized. I also use the minimized version of the Javascript file.

In the body of the HTML there is a div with the class details-content. It then renders the view that is

using the layout. There is another div with the id account-tab. This div holds the user's account details. For now just the Log Off button but in the future it will have more features. There are two anchor tags that call the function toggleAccountTab that will display the account tab. Inside the account details div there is a form with a submit but that calls the log off action.

So, there are a few things missing here. First, is the extension method IsDebug. Second, is the GameScript.js file. Third, is the CSS to style the layout. I will add them in that order. Right click the DragonsClaw project in the Solution Explorer, select Add and then Class. Name this class ExtensionMethods. Here is the code for that class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace DragonsClaw
{
    public static class ExtensionMethods
    {
        public static bool IsDebug(this HtmlHelper htmlHelper)
        {
#if DEBUG
            return true;
#else
            return false;
#endif
        }
    }
}
```

It used the compiler directive #if DEBUG to determine if we are running in debug mode. If we are I return true. If we are not running in debug mode I return false.

Now I will add the Javascript file, GameScript.js. Right click the Scripts folder in the Solution Explorer. Select Add and then JavaScript File. Name this JavaScript file GameScript. Add the following code to that file.

```
function toggleAccountTab() {
    $('#account-details').toggle();
}
```

This is a simple function. It uses jQuery to select the element with the #account-details id and toggle its visibility between hidden and visible.

Now I will add the CSS to style the HTML. Open the dragons-claw.less file and add the following CSS to it.

```
/* Account Tab CSS */
#account-tab {
    position: fixed;
```

```
    top: 0;
    right: 0;
    width: 150px;
    background-color: @highlight1;
    padding: 5px;
}

#account-details {
    display: none;
}

.account {
    text-decoration: none;
    color: #FFFFFF;
    font-size: @fontSize;
}

a.account:hover {
    color: @accent1;
}

#logoff {
    border: 0;
    background: transparent;
    color: #FFFFFF;
    font-size: @fontSize;
    font-family: 'postamtregular';
}

#logoff:hover {
    color: @accent1;
}
```

The account tab has a fixed position in the upper right hand corner of the browser. I set the width to 150px with a padding of 5px. I assign the background to the value of @highlight1. The account details element has a display value of none so it is initially hidden. Elements with the class account have no text-decoration, are white and a font-size of @fontSize. When hovering over an anchor tag with the class account the color is @accent1. The element with the id logoff has no border, background is transparent, color is white, font-size is @fontSize and the font-family is postamtregular. This CSS is overriding the behavior of a button to make it look like an anchor tag. There is also a hover selector that has the same color as the account class.

I'm going to add a couple models now. One for the player and one for creating a player. Right click the Models folder, select Add and then Class. Name this new class PlayerViewModel. Here is the code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace DragonsClaw.Models
{
    public class PlayerViewModel
    {
        public int PlayerId { get; set; }
        public bool IsAdmin { get; set; }
```

```
        public string PlayerName { get; set; }
        public string SectorName { get; set; }
        public int Credits { get; set; }
        public int Food { get; set; }
        public int Dilithium { get; set; }
        public int Ore { get; set; }
        public int Planets { get; set; }
        public int Explore { get; set; }
        public int PsionicEnergy { get; set; }
        public int Population { get; set; }
        public int Employed { get; set; }
        public int Cadets { get; set; }
        public int OffensiveTroops { get; set; }
        public int DefensiveTroops { get; set; }
        public int SpecialtyTroops { get; set; }
        public int Mercenaries { get; set; }
        public int Spies { get; set; }
        public int Raiders { get; set; }
        public int Fighters { get; set; }
        public int Bombers { get; set; }
        public int Cruisers { get; set; }
        public int Destroyers { get; set; }
        public int Dreadnaughts { get; set; }
        public int Psionists { get; set; }
        public int PsionicCrystals { get; set; }
        public int Gender { get; set; }
        public int ClassId { get; set; }
        public string ClassName { get; set; }
        public int RaceId { get; set; }
        public string RaceName { get; set; }
        public int Admirals { get; set; }
        public bool Observer { get; set; }
        public int NetWorth { get; set; }
        public int NetWorthPerPlanet { get; set; }
        public string Message { get; set; }
        public bool Success { get; set; }
        public int Stealth { get; set; }
        public int Terabytes { get; set; }
        public int Happiness { get; set; }
        public long DefenseAtHome { get; internal set; }
        public long OffenseAtHome { get; internal set; }
        public long Defense { get; internal set; }
        public long Offense { get; internal set; }
        public float DefEfficiency { get; internal set; }
        public float OffEfficiency { get; internal set; }
        public int Exploring { get; internal set; }
    }
}
```

This is just a collection of properties. There are properties for the fields in the database other than the user name and user guid. There are some additional properties. Success and Message are for handling errors. DefenseAtHome, OffenseAtHome, Defense, Offense, DefEfficiency and OffEfficiency are calculated properties for the player's army that we will get to later. IsAdmin will be used in the future as will be as will Exploring.

Right click the Models folder, select Add and then Class. Name this new class CreatePlayerModel. Here is the code.

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace DragonsClaw.Models
{
    public class CreatePlayerModel
    {
        public string Name { get; set; }
        public string SectorName { get; set; }
        public string Gender { get; set; }
        public string Race { get; set; }
        public string Class { get; set; }
        public bool Observer { get; set; }

        public SelectList RaceList { get; internal set; }
        public SelectList ClassList { get; internal set; }
        public string Email { get; internal set; }
    }
}
```

There are public read/write fields for the choices that the player makes when creating a new character in the game. There are select lists that are populated in the controller from the database that hold the races and the classes available. There is also a field for the player's email address.

Now I'm going to add a controller with two actions. An Index action and a Create action with a GET and POST. Right the Controllers folder in the Solution Explorer, select Add and then Controller. From the list the comes up choose MVC 5 Controller – Empty and click the Add button. Name the controller GameController. Here is the code.

```
using DragonsClaw.Models;
using Microsoft.AspNet.Identity;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data.SqlClient;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace DragonsClaw.Controllers
{
    public class GameController : Controller
    {
        // GET: Game
        [Authorize]
        public ActionResult Index()
        {
            if (!DataLayer.DoesPlayerExist(User.Identity.GetUserName()))
            {
                return RedirectToAction("Create");
            }

            return View(DataLayer.GetPlayer(User.Identity.GetUserName()));
        }
```

```
        [Authorize]
        public ActionResult Create()
        {
            if (DataLayer.DoesPlayerExist(User.Identity.GetUserName()))
            {
                return RedirectToAction("Index");
            }

            CreatePlayerModel model = new CreatePlayerModel();

            ExtractClassAndRace(model);
            return View(model);
        }

        private void ExtractClassAndRace(CreatePlayerModel model)
        {
            using (SqlConnection connection = new SqlConnection())
            {
                connection.ConnectionString =
ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;
                connection.Open();

                using (SqlCommand command = new SqlCommand())
                {
                    command.Connection = connection;
                    command.CommandText = "SELECT * FROM Races";

                    using (SqlDataReader reader = command.ExecuteReader())
                    {
                        List<SelectListItem> items = new List<SelectListItem>
                        {
                            new SelectListItem() { Value = "-1", Text =
"--------------------" }
                        };

                        while (reader.Read())
                        {
                            SelectListItem item = new SelectListItem
                            {
                                Text = reader.GetString(1),
                                Value = reader.GetInt32(0).ToString()
                            };

                            items.Add(item);
                        }

                        model.RaceList = new SelectList(items, "Value", "Text");

                        reader.Close();
                    }

                    command.CommandText = "SELECT * FROM Classes";

                    using (SqlDataReader reader = command.ExecuteReader())
                    {
                        List<SelectListItem> items = new List<SelectListItem>
                        {
                            new SelectListItem() { Value = "-1", Text =
"--------------------" }
                        };
```

```
                        while (reader.Read())
                        {
                            SelectListItem item = new SelectListItem
                            {
                                Text = reader.GetString(1),
                                Value = reader.GetInt32(0).ToString()
                            };

                            items.Add(item);
                        }

                        model.ClassList = new SelectList(items, "Value", "Text");
                    }
                }
            }
        }

        [Authorize]
        [HttpPost]
        public ActionResult Create(CreatePlayerModel model)
        {
            if (!ModelState.IsValid || model.Race == "-1" || model.Class == "-1" ||
model.Gender == "-1")
            {
                ExtractClassAndRace(model);
                return View(model);
            }

            DataLayer.CreatePlayer(model, User.Identity.GetUserName());

            return RedirectToAction("Index");
        }
    }
}
```

The Index action has the Authorize attribute so it requires a user be signed in. In an if statement I call a method DataLayer.DoesPlayerExist, passing in the user name of the current user. If the return value is false it redirects to the Create action. Otherwise it returns the view for the action. I will be adding the view later as well as the data layer, which is just a static class, that separates the database code from the controllers. The view takes a PlayerViewModel as its model. That is returned by the GetPlayer method of the DataLayer class.

The Create action for the GET verb has the Authorize attribute as well. In fact all of the actions that I will be implementing will have the Authorize attribute because we require the user's email in order to know who they are. I call DataLayer.DoesPlayerExist with the user's email and if it returns true I redirect to the Index action. I then create a CreatePlayerModel to pass to the view because it requires that model. I call a method ExtractClassAndRace which pulls the race and class data from the Races and Classes table and puts it into select lists.

In the ExtractClassAndRace method I create an SqlConnection. I use the ConnectionStrings property of the ConfigurationManager to get the connection string for the connection in the Web.config file. Then I open the connection. I create an SqlCommand next. The Connection property is set to the connection and the CommandText is a select statement to grab all of the races in the Races table. Next

I create a SqlDataReader and call the ExecuteReader method of the command to populate it. There is a List<SelectListItem> to create the select list with. It adds an item with a Value of -1 and a Text of dashes. While there are rows in the reader I create a SelectListItem with a Text of reader.GetString(1) which is the name of the race, the second column, and a Value of reader.GetInt32(0) which is the ID of the race. The item is then added to the list of items. The RaceList property of the model is set to a new select list using the items and the properties that indicate values and text. The reader is closed. Next I do the same thing for the Classes table.

The Create action for the POST verb checks if the ModelState.IsValid is false, or the Race property is -1, or the Class property is -1 or the Gender property is -1. If any of these are false I call the ExtractClassAndRace method passing in the model then return the view passing in the model. Once the model has been verified for valid data I call DataLayer.CreatePlayer passing in the model and the user name. I then redirect to the Index action.

As I mentioned the DataLayer class tries to separate the database code from the controllers. I probably should have made a full fledged data layer but this worked out fine. Right click the DragonsClaw project in the Solution Explorer, select Add and then Class. Name this new class DataLayer. Here is the code.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data.SqlClient;
using System.Linq;
using System.Web;
using DragonsClaw.Models;

namespace DragonsClaw
{
    public class DataLayer
    {
        public static string GetGender(int gender)
        {
            switch (gender)
            {
                case 0:
                    return "Male";
                case 1:
                    return "Female";
                default:
                    return "Non-Binary";
            }
        }

        internal static bool DoesPlayerExist(string username)
        {
            try
            {
                using (SqlConnection connection = new SqlConnection())
                {
                    connection.ConnectionString =

ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;
                    connection.Open();
```

```
                    using (SqlCommand command = new SqlCommand())
                    {
                        command.Connection = connection;
                        command.CommandText = "SELECT * FROM PLAYERS WHERE UserEmail LIKE '" +
username + "'";

                        var result = command.ExecuteReader();

                        if (result.HasRows)
                        {
                            connection.Close();
                            return true;
                        }
                    }

                    connection.Close();
                }
            }
            catch (Exception exc)
            {
                LogMessage(exc.Message, "CheckUserExists");
            }

            return false;
        }
        internal static PlayerViewModel GetPlayer(string username)
        {
            PlayerViewModel model = new PlayerViewModel();

            try
            {
                using (SqlConnection connection = new SqlConnection())
                {
                    connection.ConnectionString =

ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;
                    connection.Open();

                    using (SqlCommand command = new SqlCommand())
                    {
                        command.Connection = connection;
                        command.CommandText = "SELECT * FROM PLAYERS AS p WHERE UserEmail LIKE
'" + username + "%'";

                        var reader = command.ExecuteReader();

                        if (reader.HasRows)
                        {
                            if (reader.Read())
                            {
                                model.PlayerId = reader.GetInt32(0);
                                model.PlayerName = reader.GetString(3);
                                model.SectorName = reader.GetString(4);
                                model.Credits = reader.GetInt32(5);
                                model.Food = reader.GetInt32(20);
                                model.Planets = reader.GetInt32(6);
                                model.PsionicEnergy = reader.GetInt32(7);
                                model.Population = reader.GetInt32(8);
                                model.Employed = reader.GetInt32(9);
```

```
                        model.Cadets = reader.GetInt32(10);
                        model.OffensiveTroops = reader.GetInt32(11);
                        model.DefensiveTroops = reader.GetInt32(12);
                        model.SpecialtyTroops = reader.GetInt32(13);
                        model.Mercenaries = reader.GetInt32(14);
                        model.Spies = reader.GetInt32(15);
                        model.ClassId = reader.GetInt32(16);
                        model.RaceId = reader.GetInt32(17);
                        model.Psionists = reader.GetInt32(18);
                        model.Gender = reader.GetInt32(19);
                        model.PsionicCrystals = reader.GetInt32(21);
                        model.Fighters = reader.GetInt32(22);
                        model.Bombers = reader.GetInt32(23);
                        model.Cruisers = reader.GetInt32(24);
                        model.Destroyers = reader.GetInt32(25);
                        model.Dreadnaughts = reader.GetInt32(26);
                        model.Raiders = reader.GetInt32(27);
                        model.Dilithium = reader.GetInt32(28);
                        model.Ore = reader.GetInt32(29);
                        model.Admirals = reader.GetInt32(30);
                        model.Observer = reader.GetBoolean(31);
                        model.NetWorth = reader.GetInt32(32);
                        model.NetWorthPerPlanet = reader.GetInt32(33);
                        model.Stealth = reader.GetInt32(34);
                        model.Terabytes = reader.GetInt32(35);
                        model.Happiness = reader.GetInt32(36);
                    }
                    connection.Close();
                    model.RaceName = GetRace(model.RaceId);
                    model.ClassName = GetClass(model.ClassId);
                    model.Exploring = GetExploring(model.PlayerId);
                    CalcNetWorth(model);
                    return model;
                }
            }

            connection.Close();
        }
    }
    catch (Exception exc)
    {
        LogMessage(exc.Message, "GetPlayer");
    }

    return model;
}

private static void CalcNetWorth(PlayerViewModel model)
{
}

private static int GetExploring(int playerId)
{
    return 0;
}

private static string GetClass(int classId)
{
    try
    {
```

```
                using (SqlConnection connection = new SqlConnection())
                {
                    connection.ConnectionString =
ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;
                    connection.Open();

                    using (SqlCommand command = new SqlCommand())
                    {
                        command.Connection = connection;
                        command.CommandText = "SELECT * FROM Classes WHERE ClassId = " +
classId;

                        var result = command.ExecuteReader();

                        if (result.HasRows)
                        {
                            result.Read();
                            string Class = result.GetString(1);
                            connection.Close();
                            return Class;
                        }
                    }

                    connection.Close();
                }
            }
            catch (Exception exc)
            {
                LogMessage(exc.Message, "GetClass");
            }

            return string.Empty;
        }

        private static string GetRace(int raceId)
        {
            try
            {
                using (SqlConnection connection = new SqlConnection())
                {
                    connection.ConnectionString =
ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;
                    connection.Open();

                    using (SqlCommand command = new SqlCommand())
                    {
                        command.Connection = connection;
                        command.CommandText = "SELECT * FROM Races WHERE RaceId = " + raceId;

                        var result = command.ExecuteReader();

                        if (result.HasRows)
                        {
                            result.Read();
                            string race = result.GetString(1);
                            connection.Close();
                            return race;
                        }
                    }
```

```
                    connection.Close();
                }
            }
            catch (Exception exc)
            {
                LogMessage(exc.Message, "GetRace");
            }

            return string.Empty;
        }

        private static void LogMessage(string message, string function)
        {

        }

        internal static void CreatePlayer(CreatePlayerModel model, string username)
        {
            try
            {
                using (SqlConnection connection = new SqlConnection())
                {
                    connection.ConnectionString =
ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;
                    connection.Open();

                    using (SqlCommand command = new SqlCommand())
                    {
                        command.Connection = connection;
                        command.CommandText =
                            "INSERT INTO Players (UserGuid, UserEmail, PlayerName, SectorName,
Credits, Food, Planets, PsionicEnergy, PsionicCrystals," +
                            "Population, Employed, Cadets, OffensiveTroops, DefensiveTroops,
SpecialtyTroops, " +
                            "Mercenaries, Spies, ClassId, RaceId, Psionists, Gender, Observer)
VALUES (@UserGuid, " +
                            "@UserEmail, @PlayerName, @SectorName, 5000000, 5000, 500, 100, 0,
5000, 0, 5000, 0, 0, 0, " +
                            "0, 0, " + "@ClassId, @RaceId, 0, @Gender, @Observer)";

                        SqlParameter p = new SqlParameter
                        {
                            DbType = System.Data.DbType.String,
                            ParameterName = "@UserGuid",
                            Value = username
                        };

                        command.Parameters.Add(p);

                        p = new SqlParameter
                        {
                            DbType = System.Data.DbType.String,
                            ParameterName = "@UserEmail",
                            Value = username
                        };

                        command.Parameters.Add(p);

                        p = new SqlParameter
                        {
```

```
                DbType = System.Data.DbType.AnsiString,
                ParameterName = "@PlayerName",
                Value = model.Name
            };

            command.Parameters.Add(p);

            p = new SqlParameter
            {
                DbType = System.Data.DbType.AnsiString,
                ParameterName = "@SectorName",
                Value = model.SectorName
            };

            command.Parameters.Add(p);

            p = new SqlParameter
            {
                DbType = System.Data.DbType.Int32,
                ParameterName = "@RaceId",
                Value = int.Parse(model.Race)
            };

            command.Parameters.Add(p);

            p = new SqlParameter
            {
                DbType = System.Data.DbType.Int32,
                ParameterName = "@ClassId",
                Value = int.Parse(model.Class)
            };

            command.Parameters.Add(p);

            p = new SqlParameter
            {
                DbType = System.Data.DbType.Int32,
                ParameterName = "@Gender",
                Value = int.Parse(model.Gender)
            };

            command.Parameters.Add(p);

            p = new SqlParameter
            {
                ParameterName = "@Observer",
                Value = model.Observer
            };

            command.Parameters.Add(p);

            var result = command.ExecuteNonQuery();
        }

        connection.Close();
    }
}
catch (Exception exc)
{
    //  todo: log the exception
```

```
                DataLayer.LogMessage(exc.Message, "CreatePlayer");
            }
        }
    }
}
```

There is a utility method, GetGender, that takes an integer parameter. If the parameter is 0 it returns Male, 1 returns Female and anything else returns Non-Binary. Trying to be gender sensitive here, at least a little.

The DoesPlayerExist method is internal so it is only accessible inside the assembly. It accepts a string parameter that is the username of the user being searched. There is a try-catch block around the database code to prevent exceptions from crashing the game. When doing database queries I always follow the same process. I create an SqlConnection in a using statement so it is automatically disposed when execution leaves the block. I get the connection string from the Web.config using the ConnectionStrings property of the ConfigurationManager class. Next I open the connection. Once the connection I create the SqlCommand inside a using statement so it will be disposed. The Connection is set to the connection just created. The CommandText is then set to a SELECT that searches the Players table where the UserEmail field is like the email passed in. I capture the result of ExecuteReader in a result variable. If the result variable has rows a hit was found so I close the connection and return true. After exiting the block for the SqlCommand I close the connection. In the catch portion of the try-catch I call a method LogMessage passing in the exception's message and the name of method. Finally I return false because a player with this email has not been found.

The GetPlayer method takes a string parameter that is the user name or email of the user. The database code runs in a try-catch block so the game doesn't blow up if there is a problem executing the code. I do the SqlConnection and SqlCommand as before. For the CommandText for the SqlCommand I do a select on the Player table using UserEmail like the user name passed in. If a result was found I set the properties of the model to the fields from the database. I close the connection and call a method GetRace passing in the RaceId to assign the RaceName property of the model. I get the ClassName property by calling GetClass passing in the ClassId. I then set the Exploring property by calling the GetExploring method. I also call a method CalcNetWorth. Then I return the model. If there are no results I close the connection. I the catch I call the LogMessage method to log the error. Finally I return the empty model.

CalcNetWorth is just a method stub for now. It will be filled out later. The same is true for GetExploring.

GetClass takes an integer parameter that is the ClassId of the class to be retrieved. It creates a SqlConnection and SqlCommand to run a query. The query is a select of the Classes table where the ClassId is the parameter passed in. If there is a row I close the connection and return the ClassName field. If there is a failure I call the LogMessage method.

GetRace is basically the same as GetClass. It just works on the Races table instead of the Classes table.

I included a method stub for the LogMessage method. It will be called whenever there is a database error. It will write into a table that we haven't created yet.

The CreatePlayer method takes as parameters a CreatePlayerModel and the user name, or email, of the user. As in the previous method the database code is in a try-catch block. I then create a SqlConnection and SqlCommand as per usual. The CommandText property is an INSERT statement that inserts into the Players table. It is a parameterized query. The parameters are prefaced with an @ symbol. The parameters are for values from the model so I'm not doing string concatenation for all of the parmeters. There are some hard coded values in the insert string. They are for credits, food, planets, psionic energy, population, employed, and cadets. Next I create an SqlParamter for the @UserGuid parameter. Its value is set username parameter. It is then added to the Parameters collection of the SqlCommand. It creates a SqlParameter for @UserEmail and sets its value to the username parameter and adds it to the collection. I repeat the process for the other parameters setting them to the equivalent of the model and add them to the collection. I then capture the result of ExecuteNonQuery. In the catch block I call LogMessage passing in the message of the exception and CreatePlayer.

I'm now going to add the views for the Create and Index actions. In the Views folder right click the Game folder, Select Add and then View. Name this new view Index. For Template select Empty (Without Model). Select the Use a layout page: check box. Select the _GameLayout.cshtml file in the Shared folder for the layout. Here is the code for the view.

```
@model DragonsClaw.Models.PlayerViewModel
@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_GameLayout.cshtml";
}
@Html.Partial("_PlayerHeader", Model)
<div id="game-container">
    <div id="star-date">Star Date 23000</div>
    <div id="bot">
        <div id="bot-menu" class="nav-bar">
            <span data-spec="The Bot pane allows you to interact with the bot."
tabindex="0">Bot</span>
            <span style="float: right;">
                <a href="#!" onclick="botHide();" class="nav-link">^</a>
                <a href="#!" onclick="botShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="bot-content" class="content">
        </div>
    </div>
    <div id="high-council">
        <div id="high-council-menu" class="nav-bar">
            <span data-spec="The High Council pane contains useful information about your
sector." tabindex="0">High Council</span>
            <span style="float: right;">
                <a href="#!" onclick="highCouncilHide();" class="nav-link">^</a>
                <a href="#!" onclick="highCouncilShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="high-council-content" class="content">
        </div>
    </div>
    <div id="empire">
        <div id="empire-menu" class="nav-bar">
            <span data-spec="The Empire pane contains information about your
```

```
empire.">Empire</span>
            <span style="float: right;">
                <a href="#!" onclick="empireHide();" class="nav-link">^</a>
                <a href="#!" onclick="empireShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="empire-content" class="content">
        </div>
    </div>
    <div id="news">
        <div id="news-menu" class="nav-bar">
            <span data-spec="The News pane contains information about events in your
sector.">News</span>
            <span style="float: right;">
                <a href="#!" onclick="newsHide();" class="nav-link">^</a>
                <a href="#!" onclick="newsShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="news-content" class="content">
        </div>
    </div>
    <div id="empire-news">
        <div id="empire-news-menu" class="nav-bar">
            <span data-spec="The Empire News pane contains information about events in your
empire.">Empire News</span>
            <span style="float: right;">
                <a href="#!" onclick="empirenewsHide();" class="nav-link">^</a>
                <a href="#!" onclick="empirenewsShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="empire-news-content" class="content">
        </div>
    </div>
    <div id="exploration">
        <div id="exploration-menu" class="nav-bar">
            <span data-spec="The Exploration pane allows you to expand the borders of your
sector by exploring for new systems.">Exploration</span>
            <span style="float: right;">
                <a href="#!" onclick="explorationHide();" class="nav-link">^</a>
                <a href="#!" onclick="explorationShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="exploration-content" class="content">
        </div>
    </div>
    <div id="expansion">
        <div id="expansion-menu" class="nav-bar">
            <span data-spec="The Expansion pane grows your sector by devoting planets to
different specialties.">Expansion</span>
            <span style="float: right;">
                <a href="#!" onclick="expansionHide();" class="nav-link">^</a>
                <a href="#!" onclick="expansionShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="expansion-content" class="content">
        </div>
    </div>
    <div id="technology">
        <div id="technology-menu" class="nav-bar">
            <span data-spec="The Technology pane allows you to research technology that can
```

```
improve the effectiveness of different areas.">Technology</span>
            <span style="float: right;">
                <a href="#!" onclick="technologyHide();" class="nav-link">^</a>
                <a href="#!" onclick="technologyShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="technology-content" class="content">
        </div>
    </div>
    <div id="military">
        <div id="military-menu" class="nav-bar">
            <span data-spec="The Military pane allows you to train your military for conquest
and defense.">Military</span>
            <span style="float: right;">
                <a href="#!" onclick="militaryHide();" class="nav-link">^</a>
                <a href="#!" onclick="militaryShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="military-content" class="content">
        </div>
    </div>
    <div id="telepaths">
        <div id="telepaths-menu" class="nav-bar">
            <span data-spec="The Telepath is used to launch psychic attacks against enemy
sectors.">Telepaths</span>
            <span style="float: right;">
                <a href="#!" onclick="telepathsHide();" class="nav-link">^</a>
                <a href="#!" onclick="telepathsShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="telepaths-content" class="content">
        </div>
    </div>
    <div id="psionics">
        <div id="psionics-menu" class="nav-bar">
            <span data-spec="The Psionics pane allows you to use psionic abilities on your
sector.">Psionics</span>
            <span style="float: right;">
                <a href="#!" onclick="psionicsHide();" class="nav-link">^</a>
                <a href="#!" onclick="psionicsShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="psionics-content" class="content">
        </div>
    </div>
    <div id="war-advisors">
        <div id="war-advisor-menu" class="nav-bar">
            <span data-spec="The War Advisors pane aids in your conquest of the galaxy.">War
Advisors</span>
            <span style="float: right;">
                <a href="#!" onclick="warAdvisorHide();" class="nav-link">^</a>
                <a href="#!" onclick="warAdvisorShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="war-advisor-content" class="content">
        </div>
    </div>
    <div id="espionage">
        <div id="espionage-menu" class="nav-bar">
            <span data-spec="The Espionage pane is where you gather intelligence on other
```

```
sectors.">Espionage</span>
            <span style="float: right;">
                <a href="#!" onclick="espionageHide();" class="nav-link">^</a>
                <a href="#!" onclick="espionageShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="espionage-content" class="content">
        </div>
    </div>
    <div id="raider">
        <div id="raider-menu" class="nav-bar">
            <span data-spec="The Raiding pane allows you to take through subtly what you can't
through force.">Raiding</span>
            <span style="float: right;">
                <a href="#!" onclick="raiderHide();" class="nav-link">^</a>
                <a href="#!" onclick="raiderShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="raider-content" class="content">
        </div>
    </div>
    <div id="array">
        <div id="array-menu" class="nav-bar">
            <span data-spec="The Arrays pane allows for the construction of arrays benificial
to your empire.">Arrays</span>
            <span style="float: right;">
                <a href="#!" onclick="arrayHide();" class="nav-link">^</a>
                <a href="#!" onclick="arrayShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="array-content" class="content">
        </div>
    </div>
    <div id="platform">
        <div id="platform-menu" class="nav-bar">
            <span data-spec="The Platforms pane allows for the construction of platforms to
hinder your opponents.">Platforms</span>
            <span style="float: right;">
                <a href="#!" onclick="platformHide();" class="nav-link">^</a>
                <a href="#!" onclick="platformShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="platform-content" class="content">
        </div>
    </div>
    <div id="aid">
        <div id="aid-menu" class="nav-bar">
            <span data-spec="The Assistance pane allows you to assist other sectors in your
empire.">Assistance</span>
            <span style="float: right;">
                <a href="#!" onclick="aidHide();" class="nav-link">^</a>
                <a href="#!" onclick="aidShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="aid-content" class="content">
        </div>
    </div>
    <div id="market">
        <div id="market-menu" class="nav-bar">
            <span data-spec="The Marketplace pane allows you to buy and sell resources on the
```

```
galatic market.">Marketplace</span>
            <span style="float: right;">
                <a href="#!" onclick="marketHide();" class="nav-link">^</a>
                <a href="#!" onclick="marketShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="market-content" class="content">
        </div>
    </div>
    <div id="quests">
        <div id="quests-menu" class="nav-bar">
            <span>Quests</span>
            <span style="float: right;">
                <a href="#!" onclick="questsHide();" class="nav-link">^</a>
                <a href="#!" onclick="questsShow();" class="nav-link">v</a>
            </span>
        </div>
        <div id="quests-content" class="content">
        </div>
    </div>
</div>
```

The view takes a PlayerViewModel as its model. It renders a partial view that I will add later in the tutorial that displays a header on the page with details about the player.

There is a div with the id game-container that wraps around the entire view. The first child is a div with the id star-date. This div displays the current star date. Right now the star date is hard coded but it will be populated in the future. Next is a series of children with the same structure. There is an outer div with a unique id. They have a child with an id in the format id-menu and a class nav-bar. The nav-bar class formats the div the same. Inside that div is a span that is the title for the nav-bar. It has a data attribute of spec that displays as a tool tip when the user hovers over the text. There is another span with a float: right style to float it on the right side of the div. Inside the span are two anchor tags. The first anchor tag will collapse the next child and the second anchor tag will expand the next child. The next child has a unique id and a class content.

The bot div is for a bot that the player can run commands on. The high-council div is for information about the player's sector. The empire div is for information about the player's empire. The empire-news and player-news divs are for recent news for the empire and player. The exploration div is for exploring new territory rather than expanding via conquest. The expansion div is for building planet infrastructure for specialty roles. The technology div is for scientific research to improve efficiency of different areas. The military div is for training troops. The war-advisors tab is for military conquest. The espionage div is for, as the name implies, is for running espionage missions against opponents. The raiding div is for, as the name implies, raiding other players. The arrays div is for building arrays that affect an entire empire. They can only be created and launched by a host or vice host. The platform tab is for building platforms that are launched at enemy empires. Arrays are like platforms in that they can only be created by hosts and vice hosts. The assistance div is request aid from empire members or filling aid requests. The market place div is used to sell and trade resources. The quests div has quests that the player can complete. They are meant for observers so can get used to the game.

Right click the Shared folder under Views, select Add and then View. Name this new view_PlayerHeader. Check the Partial view check box to create a partial view. For Template select

Empty (Without Model). Here is the code for the partial view.

```
@model DragonsClaw.Models.PlayerViewModel
<div id="player-details">
    <table id="header-table">
        <thead>
            <tr id="header-title-bar">
                <th>Credits</th>
                <th>Planets</th>
                <th>Population</th>
                <th>Food</th>
                <th>Ore</th>
                <th>Dilithium</th>
                <th>Crystals</th>
                <th>Net Worth</th>
            </tr>
        </thead>
        <tr id="header-body">
            <td>@Model.Credits.ToString("N0")</td>
            <td>@Model.Planets.ToString("N0")</td>
            <td>@Model.Population.ToString("N0")</td>
            <td>@Model.Food.ToString("N0")</td>
            <td>@Model.Ore.ToString("N0")</td>
            <td>@Model.Dilithium.ToString("N0")</td>
            <td>@Model.PsionicCrystals.ToString("N0")</td>
            <td>@Model.NetWorth.ToString("N0")</td>
        </tr>
    </table>
</div>
```

The model for the view is a PlayerViewModel. It is wrapped by a div with the id player-details. Since this is tabular data I use a table to display it. The table has an id of header-table. In the head of the table there are columns for the player's credits, planets, population, food, ore, dilithium, psionic crystals and networth. In the body there are columns that write the appropriate value of the model. I format the numbers with no decimals and with thousands separators in the user's regional settings using ToString(“N0”).

In the Views folder right click the Game folder, Select Add and then View. Name this new view Create. For Template select Empty (Without Model). Select the Use a layout page: check box. Select the _GameLayout.cshtml file in the Shared folder for the layout. Here is the code for the view.

```
@using DragonsClaw.Models
@model DragonsClaw.Models.CreatePlayerModel

@{
    Layout = "~/Views/Shared/_GameLayout.cshtml";
    ViewBag.Title = "Create";
    List<SelectListItem> genderList = new List<SelectListItem>();
    SelectListItem item = new SelectListItem() { Value = "-1", Text = "--------------------" };
    genderList.Add(item);

    item = new SelectListItem() { Value = "0", Text = DataLayer.GetGender(0) };
    genderList.Add(item);

    item = new SelectListItem() { Value = "1", Text = DataLayer.GetGender(1) };
    genderList.Add(item);
```

```
    item = new SelectListItem() { Value = "2", Text = DataLayer.GetGender(2) };
    genderList.Add(item);

    SelectList genders = new SelectList(genderList);
}

<div class="content-container">
    <div class="details-content">
        <h2>Create Sector</h2>
        <div id="create-container">
            <form id="create-form" action="/Game/Create" method="post">
                @Html.HiddenFor(m => m.Email)
                <div class="form-field">
                    <div class="create-label">Ruler Name:</div>
                    @Html.TextBoxFor(m => m.Name, new { @class = "create-field" })
                </div>
                <div class="form-field">
                    <div class="create-label">Sector Name:</div>
                    @Html.TextBoxFor(m => m.SectorName, new { @class = "create-field" })
                </div>
                <div class="form-field">
                    <div class="create-label">Gender:</div>
                    @Html.DropDownListFor(m => m.Gender, new SelectList(genderList, "Value",
"Text"), new { @class = "create-field" })
                </div>
                <div class="form-field">
                    <div class="create-label">Race:</div>
                    @Html.DropDownListFor(m => m.Race, Model.RaceList, new { @class = "create-
field" })
                </div>
                <div class="form-field">
                    <div class="create-label">Class:</div>
                    @Html.DropDownListFor(m => m.Class, Model.ClassList, new { @class =
"create-field" })
                </div>
                <div class="form-field">
                    <div class="create-label">@Html.CheckBoxFor(m => m.Observer) Observer</div>
                    Observers cannot attack or be attacked. The same is true for espionage and
raiding.
                </div>
                <input id="submit-create" type="submit" value="Create" />
            </form>
        </div>
    </div>
</div>
```

The model for the view is a CreatePlayerModel. The Layout for the view is the _GameLayout that we specified when creating the view. It creates a SelectList genderList. The first item has a value of -1 and a text property of dashes. The other items are populated with a value and a text property of the value returned by calling DataLayer.GetGender with the value.

There is a wrapper div with the class content-container. Inside of that is a div with the class details-content. There is a h2 tag with the text Create Sector. The next child is a div with the id create-container. There is next a form with the id create-form, the action /game/create which calls the create action of the Game controller and the method is post. There is a hidden form field Email. The fields of the form are wrapped in divs with the class form-field. There are labels for the forms with the class

create-label. The actual fields have the class create-field. The fields are either TextBoxFor or DropDownListFor. For the DropDownListFor for the Gender property I use the genderList that I created. For Race I use the RaceList property of the model. Similarly for the Class property I use the ClassList property of the model.

The next thing that I'm going to tackle in this tutorial is adding the CSS for the view. I will do it in stages. First, add the following CSS to the dragons-claw.less file at the end. I placed a comment at the head to mark that it is specific to the Index view.

```
/* Index View */
span[data-spec] {
    position: relative;
    cursor: help;
}

span[data-spec]:hover::after,
span[data-spec]:focus::after {
    content: attr(data-spec);
    position: absolute;
    left: 0;
    top: @largeFontSize;
    min-width: 300px;
    border: 1px solid #AAAAAA;
    border-radius: 10px;
    background-color: #FFFFD5;
    padding: 10px;
    color: #000000;
    font-size: 14px;
    z-index: 1;
    box-shadow: rgba(0, 0, 0, 0.25) 5px 5px 4px 1px;
    text-shadow: none;
    text-align: left;
}

.nav-bar {
    background-color: #3830EB;
    background: linear-gradient(@highlight1, #0B04BE, #070373);
    font-size: @largeFontSize;
    padding: 5px 10px 5px 10px;
    margin: 0;
    border-left: 5px ridge @grey3;
    border-right: 5px ridge @grey3;
    margin-right: -2px;
    text-shadow: 5px 5px 10px black;
}

.nav-bar a {
    color: white;
    text-decoration: none;
}

.content {
    position: relative;
    display: none;
    width: 800px;
}

.nav-link {
```

```
}

#game-container {
    border-left: 1px solid white;
    border-right: 1px solid white;
    border-bottom: 1px solid white;
    margin-top: 63px;
}

#star-date {
    padding-top: 15px;
    padding-bottom: 15px;
    text-align: center;
    border: 5px ridge @grey3;
    margin-right: -3px;
}
```

There is a selector for spans with a data-spec attribute. I set their position to relative for using ::after. I also set the cursor to the help cursor. There is then selectors for the same span but for hover:after and focus:after. The focus:after works because I gave the spans a tab-index property. The content is set to the data in the data-spec attribute. The position is absolute with a left of 0 which will line it up on the left side of the span. The top is @largeFontSize which gives a bit of padding at the top. The min-width is set to 300 so that the bubbles will all be the same width. I give the bubble a gray border with rounded corners. The color of the text is black. The font-size is 14 pixels. It has a z-index of 1 so it will float on top of the span. I give it a box-shadow and text-align left.

The panel bars have a fall back color of 3830EB in the case that the player's browser does not support linear gradients. I then create a linear-gradient with 3 stops. The first stop is the highlight1 color, which is a blue, and the other stops are shades of blue. They have a font size of largeFontSize. They have a top and bottom padding of 5px and a left and right of 10px. They have 0 margin. The left and right border are 5px wide with a ridge and a color of grey3. I override the right margin by -2px for the borders. I give the text a shadow in black. Any anchors in a nav-bar are white with no text decoration.

The divs with a class content have a position of relative. They are set initially to display none so they are not visible. They will be made visible when the expand button is clicked and collapsed when the collapse button is clicked. They have a width of 800px, which is the width of parent container.

The game-container id has left, right and bottom borders with a size of 1px, are solid and white. There is a top margin of 63px. It is there because there will be a fixed header and if we scroll up without the margin it looks bad.

The div with the id star-date has a top and bottom padding of 15px. The text inside of the element is centered. It has a border of 5px in grey3 with a right margin of -3px so that the borders line up correctly.

That is it for the Index view. Now I will add the code for the Create view. I added it under the comment Forms. Add the following CSS to the dragons-claw.less file.

```
/* Forms */
```

```
#create-container {
    text-align: center;
    width: 610px;
}

#create-form {
    width: 600px;
    background: #555;
    border: 5px ridge white;
    text-align: center;
}

.form-field {
    display: flex;
    flex-direction: row;
    padding: 5px;
    margin: 5px;
    font-size: @fontSize;
}

.create-label {
    width: 200px;
    color: white;
    text-align: left;
}

.create-field {
    width: 300px;
    color: white;
    background-color: #070373;
    font-family: 'postamtregular';
    font-size: @fontSize;
}
```

The element with the id create-container is 610px wide and a text-align property of center. The element with the id create-form has a width of 600px and a background color of grey. It has a solid border that has a size of 5px with a ridge style and is white. It also has a text-align property of center.

The elements with the class form-field have a display mode of flex and a flex-direction of row. They have a margin and padding of 5px. They also have a font-size of fontSize. The elements with a class of create-label have a width of 200px, color white and text-align left. The elements with a class of create-field have a width of 300px, are color white, a background color that is a dark blue, the font-family is the postamtregular font with a font-size of fontSize. I specified the font-family here because controls don't inherit the same way other HTML elements do.

There is some CSS for the _PlayerHeader partial view to format the table nicely. Adding the following code to dragons-claw.less.

```
/* Player Header */

#player-details {
    position: fixed;
    top: 0;
    z-index: 1;
}
```

```
#header-table {
    width: 800px;
    font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans', Arial,
sans-serif;
    font-size: @fontSize;
    border-collapse: collapse;
}

#header-table th, #header-table td {
    border: 1px solid white;
}

#header-title-bar {
    background-color: @highlight1;
    text-align: center;
}

#header-body {
    text-align: right;
}
```

The header is fixed at the top of the screen. The table has a width of 800px. There is then a font-family that falls back to sans-serif if none of the other fonts are available. It has a border-collapse of collapse so it is only single lines. The table has a border of 1px that is solid and white. The background of the title bar is highlight1 and the text is centered. The text in the body is right aligned.

I added a little jQuery to the GameScript file to prevent the default action when the user clicks one of the expand/collapse links. Add the following code to the GameScript file.

```
$(document).ready(function () {
    $('a.nav-link').click(function (e) {
        e.preventDefault();
    });
});
```

It runs when the document loads. Any anchor with the nav-link class will not have the default action fire on click.

You should be able to build and run now. When you go to the game you will be presented with the create sector page. After creating a sector you should be redirected to the game page.

That's a lot to digest in one sitting so I'm going to wrap the tutorial up here. I encourage you to follow my blog, follow me on Facebook @GameProgrammingAdventure or on Twitter @LadyAmethyst416 for the latest news on my tutorials.

I wish you the best in your Game Programming Adventures!