WELCOME TO THE WONDERFUL WORLD OF EAMON CS

PREFACE

[Note: this document is a work in progress]

This document is intended to describe the various facets of Eamon CS. The real documentation will be a formal set: full source code commenting, a Player's Manual, a Dungeon Designer's Manual and a Class Reference. It will take quite a while to put all that together, so in the meantime, hopefully, this will suffice.

Eamon CS (ECS) is a port of the Eamon roleplaying game to the C# programming language. It is the production version of a prior system called Eamon AC (EAC), an ANSI C prototype intended to extract Eamon out of BASIC. If you have EAC on your system, you should uninstall it before using ECS, as it is now obsolete.

This system is a hybrid of Donald Brown's Classic Eamon and the most modern BASIC Eamon available, Frank Black Production's Eamon Deluxe. It is also directly based on the EAC prototype. Eamon CS borrows liberally from and bears a strong resemblance to all these sources in various areas.

The game has evolved rapidly over its short lifespan. First, it moved to a plugin-based architecture unique to this branch of Eamon, then into the various flavors of Unix, and most recently into the Android mobile device space. The game engine has improved with each one of a small, but growing collection of custom-built adventures created along the way. The different facets of Eamon CS share a common code base. Where necessary it will be distinguished as Eamon CS Desktop for traditional workstations and Eamon CS Mobile for mobile devices.

PREREQUISITES

As mentioned in the README.md file, Eamon CS has no formal installer. There are good points and bad points to this. On the one hand, it is a simple matter of unzipping the Eamon-CS-master.zip file downloaded from GitHub to set up for play (if you did that, as opposed to doing a Git Clone). Also, it means there is no installer to maintain, which frees up time for other things like engine enhancements or developing adventures.

A disadvantage to this approach is the ECS developer/gamer is responsible for installing all system prerequisites. Otherwise, ECS will not run when any of the .bat or .sh QuickLaunch files are double-clicked. Typically, this is manifested by a console window popping up and then disappearing quickly, caused by a missing dotnet.exe program. If you find that you cannot get Eamon CS to run on Windows, your first order of business should be to download and install the following files (in order):

Programmers who want to play, step through, or develop the source code:

Microsoft .NET Framework 4.8 SDK
Microsoft .NET Core 2.2 SDK
Microsoft Visual Studio Community 2019

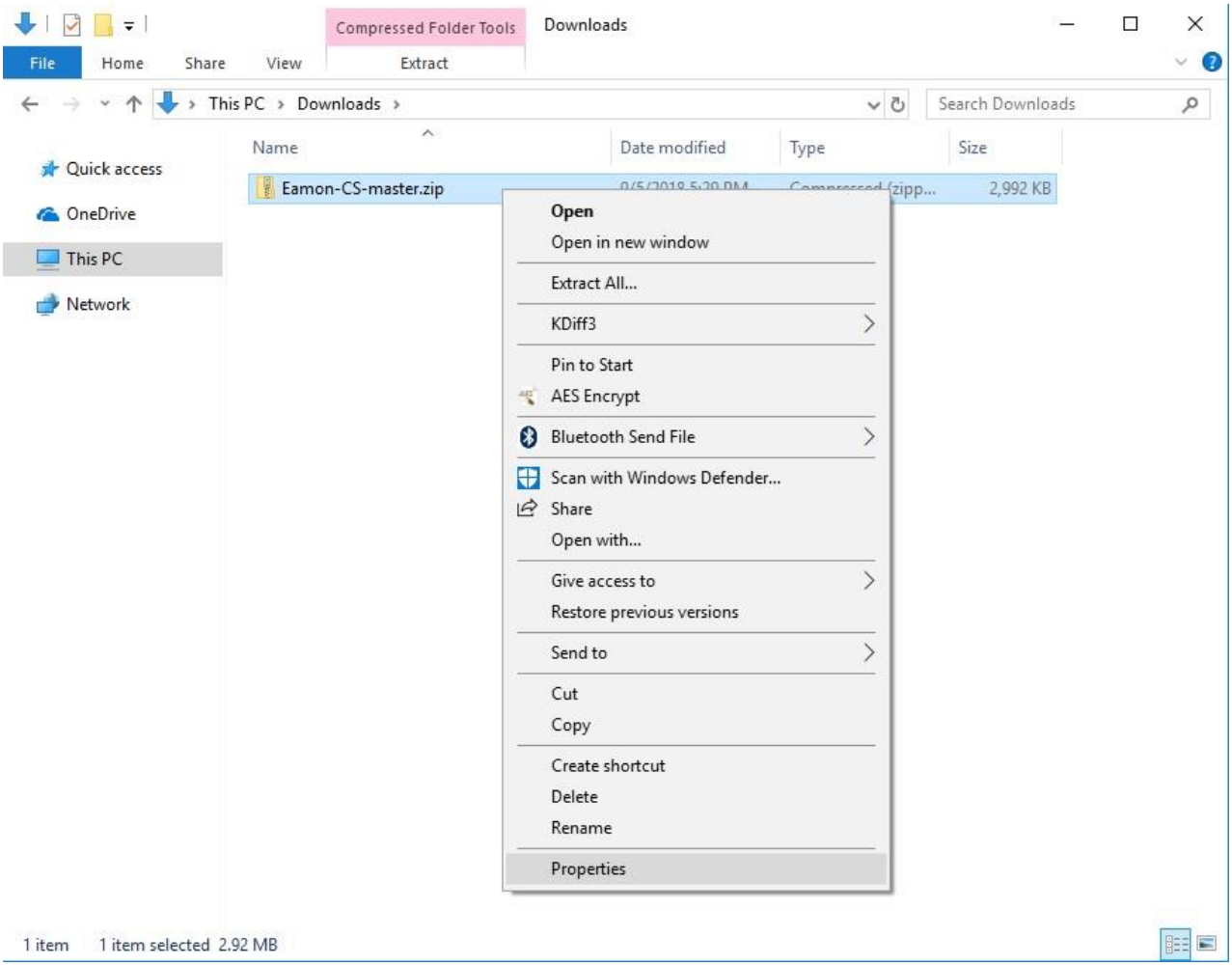
- Installer option: .NET desktop development
- Installer option: Mobile development with .NET

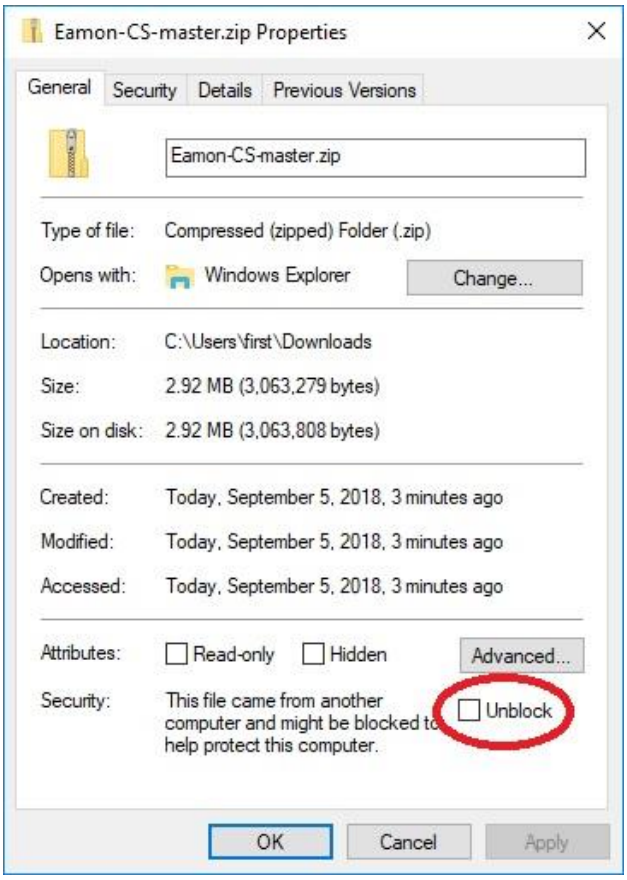
Gamers who want to play:

Microsoft .NET Framework 4.8
Microsoft .NET Core 2.2

Note: Eamon CS should be fully compatible with new versions of .NET Core. If you wish to run the system using a more recent version of .NET Core, you must download VS 2019+ and recompile the Plugin Manager program to target it. The newer .NET Core must also be installed on your system. See the Post-Installation Topics section for more details.

A second disadvantage on Windows leads to a useful tip regarding the downloaded .zip file. You should always right-click on the .zip file and select "unblock" to avoid the security warnings, as shown below:

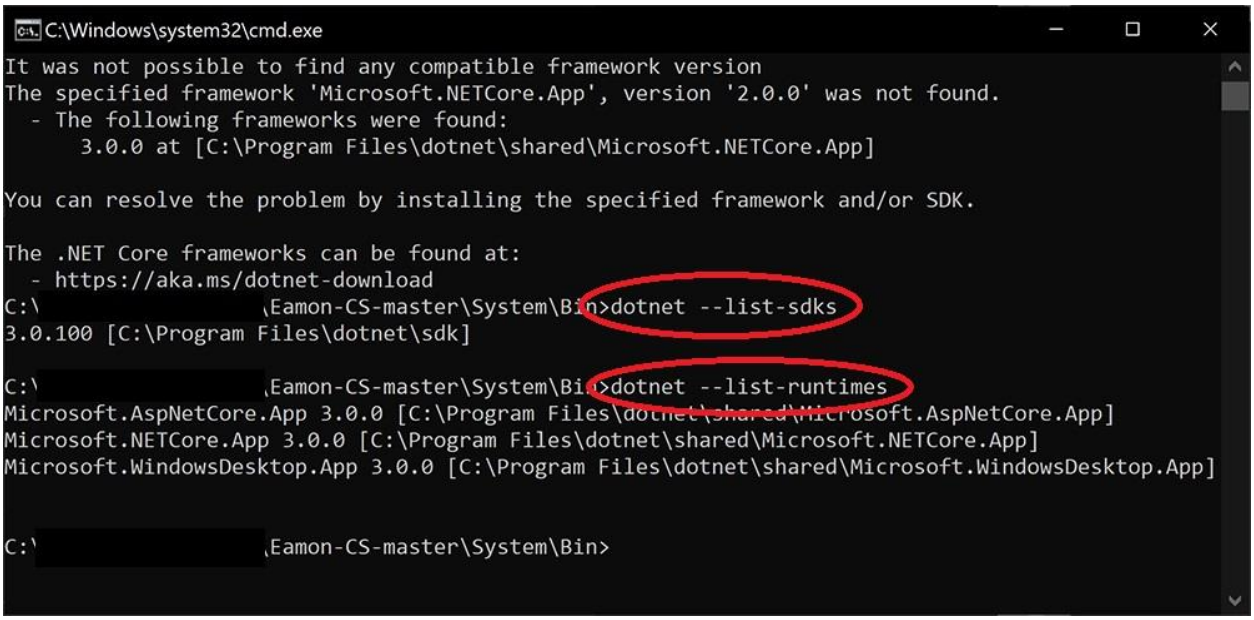

Compressed Folder Tools
Downloads
File
Home
Share
View
Extract
This PC › Downloads ›
Search Downloads
Quick access
OneDrive
This PC
Network
Name
Date modified
Type
Size
Eamon-CS-master.zip
2,992 KB
Open
Open in new window
Extract All...
KDiff3
Pin to Start
AES Encrypt
Bluetooth Send File
Scan with Windows Defender...
Share
Open with...
Give access to
Restore previous versions
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties
1 item
1 item selected 2.92 MB



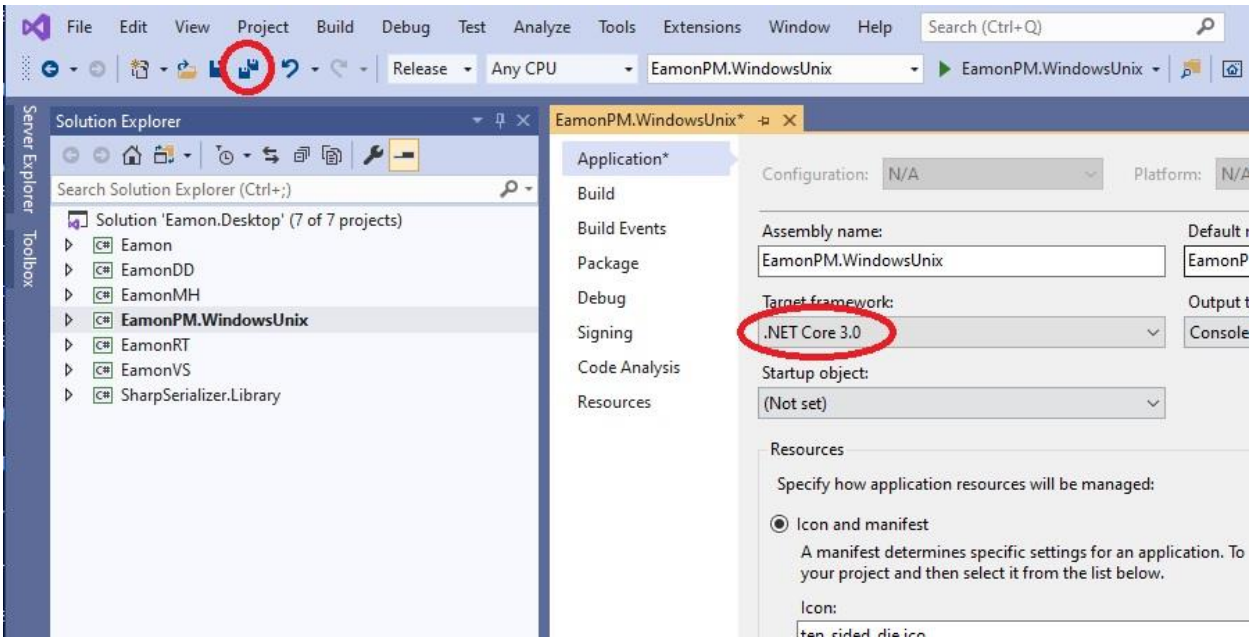
The prerequisites on Unix are similar: a version of Mono that is .NET Framework 4.7.2+ and .NET Core 2.X compatible. The means to satisfy these requirements vary from platform to platform. Eamon CS on Unix should only be pursued by those who are confident enough to see it through via their own devices. The game runs correctly on Ubuntu Linux and FreeBSD, and may also work on other Unix variants. However, due to input-related issues, OS X requires a formal port to be done.

POST-INSTALLATION TOPICS

The binaries for Eamon CS are compiled to use .NET Core 2.X, but the system should be fully compatible with new versions of .NET Core that conform to .NET Standard 2.0. If you have a newer version of .NET Core on your system and don't want to download 2.X, you can recompile the EamonPM.WindowsUnix project to target it. To do this, you must first install Visual Studio 2019+, as described in the Prerequisites section. The use of Visual Studio is described in the section called Using The Visual Studio Debugger. The following screenshot shows how to determine which versions of .NET Core are present on your system:



To recompile the EamonPM.WindowsUnix project, load the Eamon.Desktop solution in Visual Studio. Right-click on the project and choose Properties. Select the appropriate version of .NET Core in the Target Framework drop-down list on the Application tab, then click the Save button:



After doing this, right-click on the EamonPM.WindowsUnix project and choose Clean. Then right-click on the project and select Build. If you receive no errors, Eamon CS should now work with your installed .NET Core.

PROGRAMS, LIBRARIES AND PLUGINS

The plugin-based architecture used by Eamon CS extends to the Main Hall, the Dungeon Designer, and all adventures. A Plugin Manager program specific to either Desktop or Mobile environments manages them:

System\Bin\EamonPM.WindowsUnix.dll	Eamon CS Windows/Unix Plugin Manager
System\Bin\EamonPM.Android-Signed.apk	Eamon CS Android Plugin Manager
System\Bin\EamonDD.dll	Eamon CS Dungeon Designer Plugin
System\Bin\EamonMH.dll	Eamon CS Main Hall Plugin
System\Bin\EamonRT.dll	Eamon CS Adventure Plugin
System\Bin\EamonVS.dll	Eamon CS/Visual Studio Automation Library
System\Bin\Eamon.dll	Eamon CS Library
System\Bin\Polenter.SharpSerializer.dll	SharpSerializer.Library

Additionally, you have some adventures:

System\Bin\TheBeginnersCave.dll	Eamon CS Adventure Plugin
System\Bin\BeginnersForest.dll	Eamon CS Adventure Plugin
System\Bin\TheTrainingGround.dll	Eamon CS Adventure Plugin
System\Bin\TheSubAquanLaboratory.dll	Eamon CS Adventure Plugin
System\Bin\ARuncibleCargo.dll	Eamon CS Adventure Plugin
System\Bin\StrongholdOfKahrDur.dll	Eamon CS Adventure Plugin
System\Bin\TheTempleOfNgurct.dll	Eamon CS Adventure Plugin
System\Bin\WrenholdsSecretVigil.dll	Eamon CS Adventure Plugin
System\Bin\ TheVileGrimoireOfJaldial.dll	Eamon CS Adventure Plugin
System\Bin\ LandOfTheMountainKing.dll	Eamon CS Adventure Plugin

Conceptually, each plugin is a discrete program. It exposes a Program class with a Main method that takes a collection of arguments, much like any C-based program. The difference is that the Plugin Manager is what "executes" the plugin, not the calling C# environment. At the bottom of the software stack, you have Polenter.SharpSerializer.dll which handles loading and saving of the game's textfiles and Eamon.dll which holds code common to all ECS plugins. EamonRT.dll contains the vanilla game engine used by all non-customized adventures. For customized adventures, the game plugin (e.g., TheBeginnersCave.dll) contains custom code specific to that game, built by leveraging EamonRT.dll. It is a "modded" version of the game engine.

Plugins can share any logic with this implementation. For example, an interesting idea would be to create a "campaign library" that contains shared code for many derived adventures.

The source code for Plugin Managers and system plugins resides in the appropriate directory under System; for adventures, it is in the adventure-specific directory under Adventures.

The Main Hall textfiles reside in System\Bin, making them easily accessible to all plugins. Game-specific textfiles (both original and save game) reside under Adventures in their respective game directories.

Here is a final note on the architectural differences between ECS Desktop and Mobile. For Desktop, the plugins reside in the System\Bin directory and are loaded by EamonPM.WindowsUnix.dll only when needed (and reused once loaded). In contrast, when building EamonPM.Android-Signed.apk, all plugins are statically linked in, and all textfiles are embedded, producing a monolithic application. When delivering the .apk onto the mobile device, the textfiles copy (when appropriate) to the device's file system - the plugins remain part of the application. In spite of this, the plugin managers are very similar internally.

BATCH FILES

The plugins take a variety of command-line parameters (described below). However, to get you up and running quickly, there is a QuickLaunch folder. (Since there is no formal ECS installer, you may want to create a shortcut to it on your desktop manually). Inside this folder is a set of batch files that you can run directly. The batch files organize into sub-folders based on the underlying plugin they invoke:

EamonDD\EditAdventures.bat	Edit the flat Adventures database
EamonDD\EditCatalog.bat	Edit the adventure category Catalog
EamonDD\EditCharacters.bat	Edit the Characters file
EamonDD\EditContemporary.bat	Edit the Contemporary adventures category
EamonDD\EditFantasy.bat	Edit the Fantasy adventures category
EamonDD\EditSciFi.bat	Edit the Sci-Fi adventures category
EamonDD\EditTest.bat	Edit the Test adventures category
EamonDD\EditWorkbench.bat	Edit the Developer's Workbench
EamonDD\EditWorkInProgress.bat	Edit the Work-In-Progress adventures category
EamonDD\EditARuncibleCargo.bat	Edit A Runcible Cargo
EamonDD\EditBeginnersForest.bat	Edit Beginner's Forest
EamonDD\Edit[AdventureName].bat	Edit [AdventureName]
EamonDD\LoadAdventureSupportMenu.bat	Load Adventure Support Menu
EamonMH\EnterMainHallUsingAdventures.bat	Enter the Main Hall using a flat adventure database[1]
EamonMH\EnterMainHallUsingCatalog.bat	Enter the Main Hall using a hierarchical adventure database[1]
EamonRT\ResumeARuncibleCargo.bat	Resume A Runcible Cargo[2]
EamonRT\ResumeBeginnersForest.bat	Resume Beginner's Forest[2]
EamonRT\Resume[AdventureName].bat	Resume [AdventureName][2]

[1] Run these batch files to create a new character or send an existing one into the Main Hall. The only difference between the two batch files is the nature of the adventure database loaded (the same characters will be available regardless).

[2] Run these batch files to return to an in-progress adventure. Use the RESTORE command to pick up where you left off.

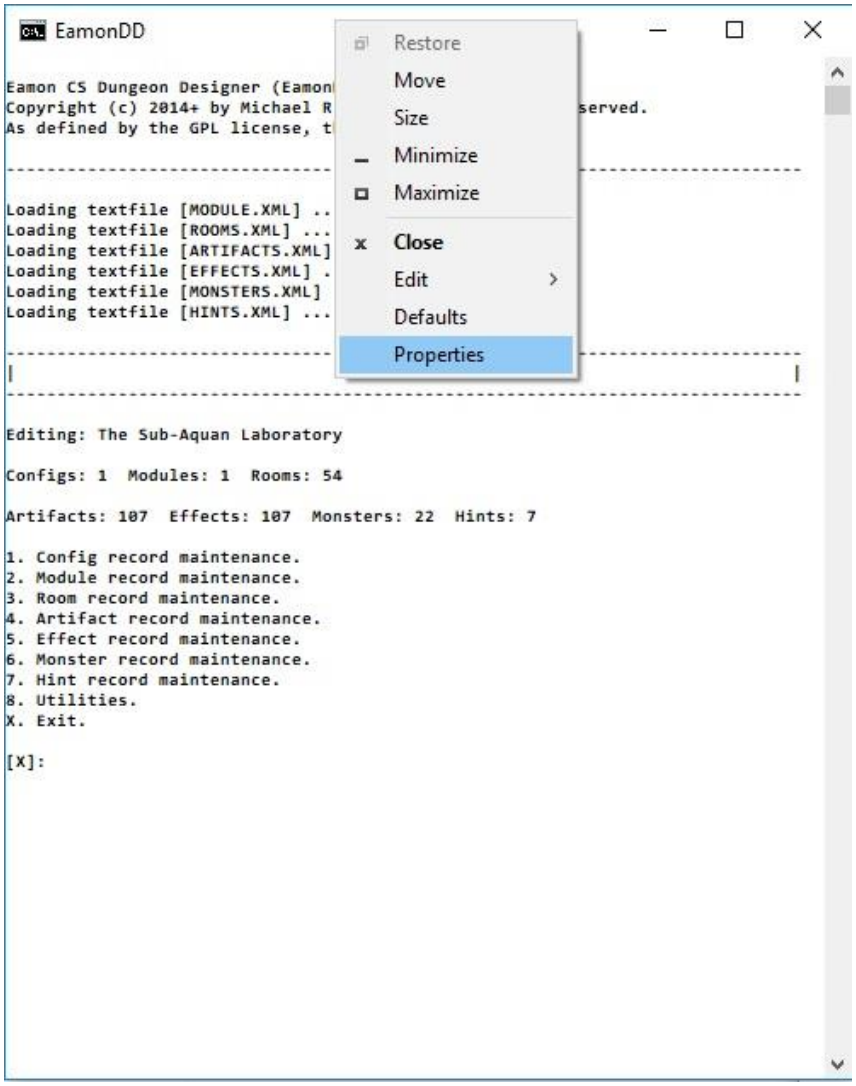
You can study the batch files to see how the system launches various programs. You can also create new batch files using these as templates if you decide to write adventures or run the system in a non-default manner.

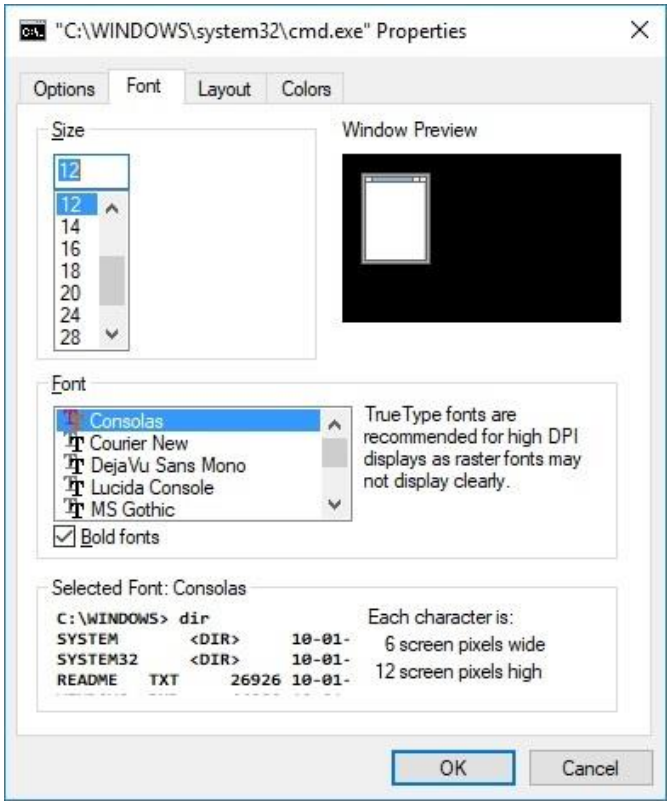
Eamon CS Mobile mirrors this QuickLaunch hierarchy using a series of ListViews to provide a similar experience.

SUGGESTED GAMEPLAY SETTINGS

Eamon CS runs as a set of 64-bit Console applications. It became apparent during development that the default settings for Console windows are not ideal when playing a text-based game. The system tries to adjust the window size to an optimal setting, but you may wish to experiment with the other settings to see what appeals to you. Strangely enough, I've found aesthetics make a big difference in the quality of the gameplay as well as the ease of use for tools like EamonDD. This topic is highly subjective, but the following ideas produced some interesting results:

1. When running ECS, right-click on the Console window title bar and choose properties. Windows 10 provides the following settings; the properties available on other versions may differ.





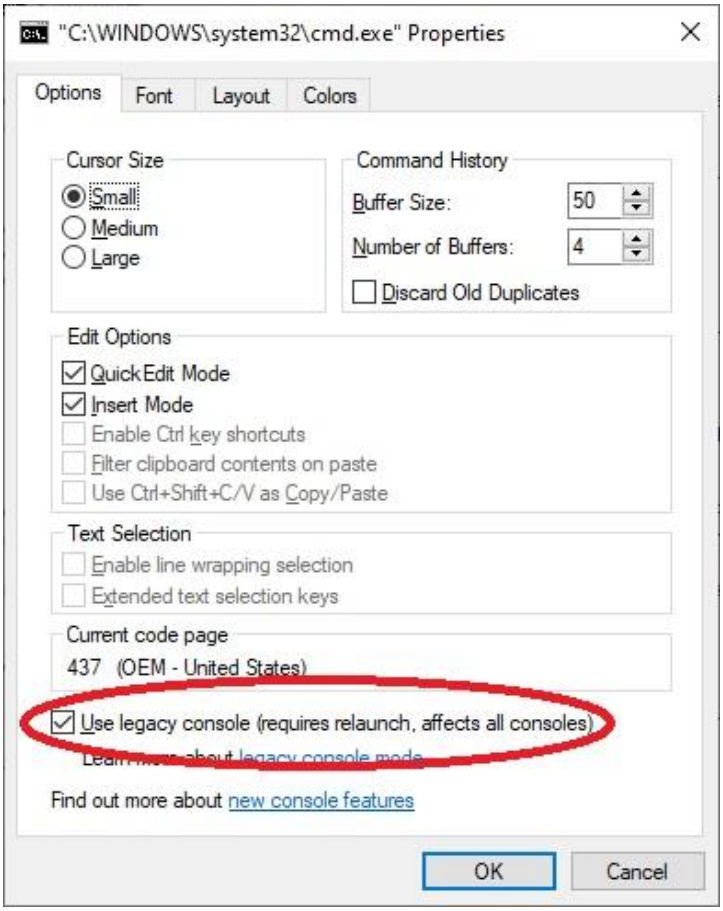
2. Font Tab -
   Size: 12-16
   Font: Consolas or Deja Vu Sans Mono
   Bold: Optional

3. Layout Tab -
   Screen Buffer Size –
      Width: 80
      Height: 9999
   Window Size –
      Width: 80
      Height: 50 (you can adjust this up or down based on screen resolution)

4. Colors Tab -
   Screen Text –
      Selected Color Values –
         Red: 0
         Green: 255
         Blue: 0
   Screen Background -
      Selected Color Values -
         Red: 0
         Green: 0
         Blue: 0

Eamon CS tries to set the values on the Layout Tab to those shown above, so you shouldn't need to change them. The Screen Buffer Width and Window Width should always be 80 columns. The Screen Buffer Height is strongly suggested to be 9999. The Screen Text shown above is Green with a Black background and has a retro-vibe to it, like an Apple II monitor. If you prefer amber the RGB values [255,165,0] might work for you. As time has passed, I settled on a white background with black text, as shown in the various images in this document. Once you have the settings you're happy with, click OK and they should persist across ECS programs.

The newest versions of Windows (10+) contain a rewritten command console program (CMD.exe). It seems to have bugs that produce intermittent cursor artifacts in Eamon CS. If you are experiencing this issue, you might try running the command console in legacy mode. Right-click on any Eamon CS console window, choose Properties, and then the Options tab. Ensure the Use Legacy Console checkbox is checked:



MISCELLANEOUS GAMEPLAY NOTES

The parser of the game has been enhanced to allow more flexible input. You can enter stuff like "PUT my sword INSIDE my backpack" or "OPEN that jewelry box" or "ATTACK the rats" or "GIVE the spices TO the hermit". The standard [VERB] [subject1] [subject2] pattern of Eamon Deluxe remains, but ECS will discard the articles before processing further. Even though it is more typing, some players may find it more immersive. Of course, you can always use the compact syntax if you prefer.

The system has been enhanced to support a variety of container types. You can store contents in, on, under, or behind depending on the container. Under and Behind Containers are similar to In Containers, but cannot be opened or closed and have no key. On Containers have special behavior, the contents are exposed and can be manipulated using any standard command.

The game engine contains a few new commands, summarized below:

1. "INFO" gives information about the played adventure.

2. "SETTINGS" allows you to control a variety of general and game-specific configuration options. You can change your settings at any point. It is a good idea to use this command when a game is started to see what's available to you.

3. "HINTS" gives you a list of hints for the current adventure. You should always carefully read EAMON CS 1.6 GENERAL HELP and EAMON CS 1.6 NEW COMMANDS as these hints may be subtly different than their Eamon Deluxe counterparts. Consider the following:

4. "GO" [door] allows you to pass through a free-standing non-directional door (e.g., the overgrown path in Test Adventure).

5. "INVENTORY" [container] is a parallel to "INVENTORY" [monster]. This command lists the container's contents, and typically only works with In Containers and On Containers.

6. EXAMINE or LOOK [in|on|under|behind] [container] can also be used to list the contents of a container. These verbs are sensitive to the preposition used and will display the appropriate content list if any. There are a variety of prepositional synonyms available. If you use a preposition unrelated to containment (or none at all), it will result in the default behavior where the description is displayed.

QUICK ARCHITECTURAL OVERVIEW

The entire Eamon CS system was written from scratch and aggressively exploits the C# language. Not just a port, this is an expansive toolkit used to build games based on the Eamon ruleset. It has more in common with traditional Interactive Fiction systems like TADS or Inform. Eamon CS games are built by sub-classing existing classes in any library/plugin, but especially Eamon.dll or EamonRT.dll, and overriding default behavior where needed. You can subclass every class in the system, and generic improvements made to individual adventures can be pushed back into the base framework for use by all games. As time goes on, the system will grow in flexibility and power.

The textfile format used by Eamon CS deserves an explanation. The game produces ASCII XML files that represent C# object graphs, a radical data storage paradigm shift. These files also play a critical role in the development of customized adventures. Take a look in the MONSTERS.XML file in Test Adventure (a non-customized adventure), and you will see the serialized class is Eamon.Game.Monster (from Eamon.dll). But in The Beginner's Cave, you will see it's TheBeginnersCave.Game.Monster (from TheBeginnersCave.dll). These are two different classes, the second is a subclass of the first, and implements custom code that is explicitly used by The Beginner's Cave. Now compare the batch file EditTestAdventure.bat, which contains the command-line argument "-pfn EamonRT.dll" with EditTheBeginnersCave.bat, which includes "-pfn TheBeginnersCave.dll". In the first, the flag instructs EamonDD to load the base engine. In the second, it loads TheBeginnersCave.dll and uses any customized classes it finds (like the Monster class) when creating new objects. This whole topic is fairly complex and begs to be part of a formal document set, but at least this gives you an idea of how it works.

With BASIC Eamons that use files containing just data, you can bypass the Dungeon Designer program altogether and build these files by hand. You should avoid this practice in Eamon CS due to the specialized textfile format. The EamonDD program is instrumental in producing textfiles of the correct format, and you should rely on it to do so. However, you can still manually tweak the textfiles with your favorite editor. Just be very careful not to violate the file format, as the Polenter.SharpSerializer.dll library can be picky.

The EamonRT.dll base adventure runtime takes the form of a Finite State Machine (FSM). For those unfamiliar, this is a technique used to model complex program behavior. The main game loop is 15 lines long; all complex behavior resides in subclasses of the State class. Player commands derive from the Command class, which is itself derived from State. Customized adventures will almost always subclass various States or Commands to provide specialized behavior. It is also straightforward to create new States or Commands and link them into the FSM if needed. Finally, if you look through the code, you will see that special care has been taken to avoid the use of GOTO except in specific circumstances (goto Cleanup). Doing this gives the game engine a clean, deterministic quality.

USING THE VISUAL STUDIO DEBUGGER

Eamon CS was built using Visual Studio 2019, a powerful, modern integrated development environment (IDE) used by programmers worldwide. It can be obtained using a link in the Prerequisites section. There are many tutorials on the web that go into detail on using Visual Studio; that topic is outside the scope of this document. If you do some research on your own, make sure you focus on using the debugger, which will be the point of this section. The IDE can be used to develop new games, modify existing games, and to step through the source code at all levels to better understand its inner workings. A few more notes:

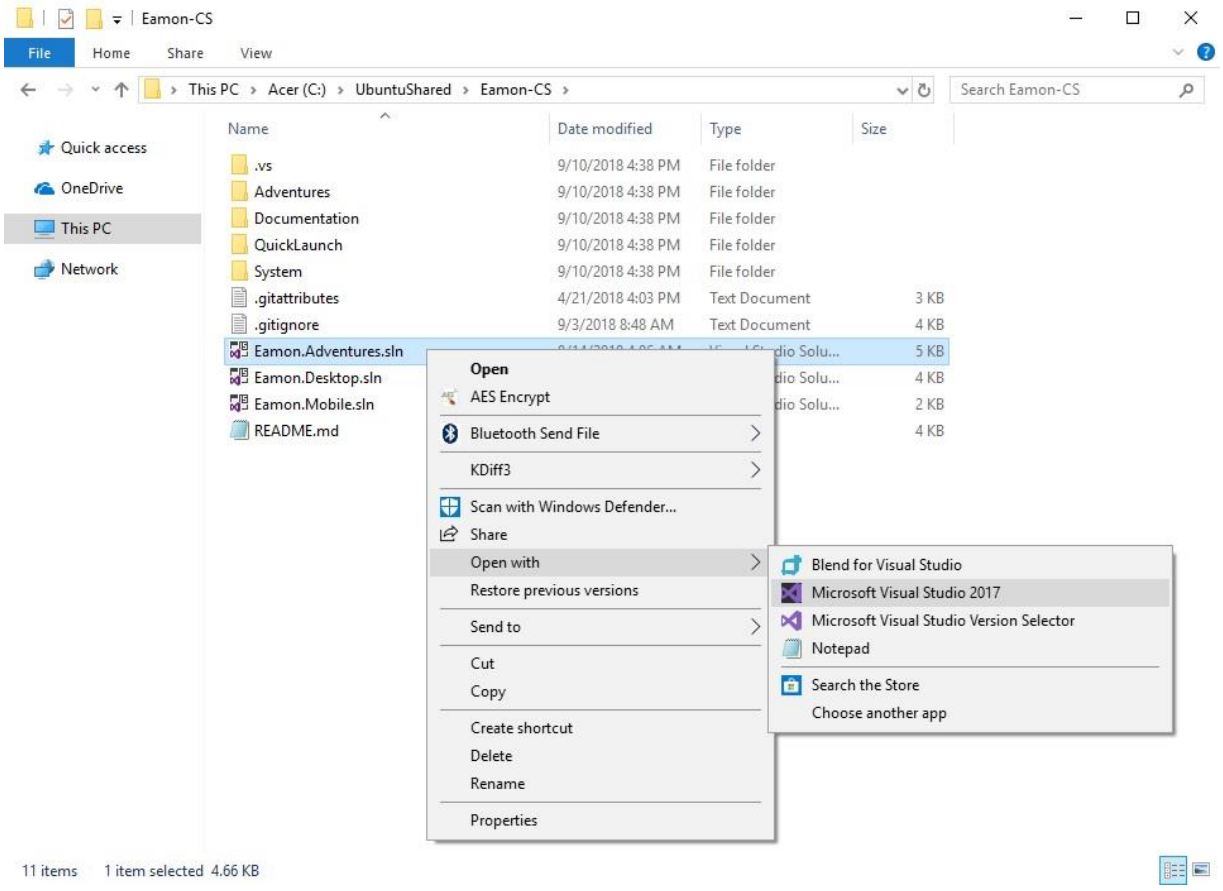
- The screenshots depict VS 2017, but newer versions of Visual Studio should be functionally equivalent.
- This discussion assumes VS 2019 on Windows; if you're using a different development environment or a different OS, the steps may vary.

To step through the Eamon CS source code, do the following:

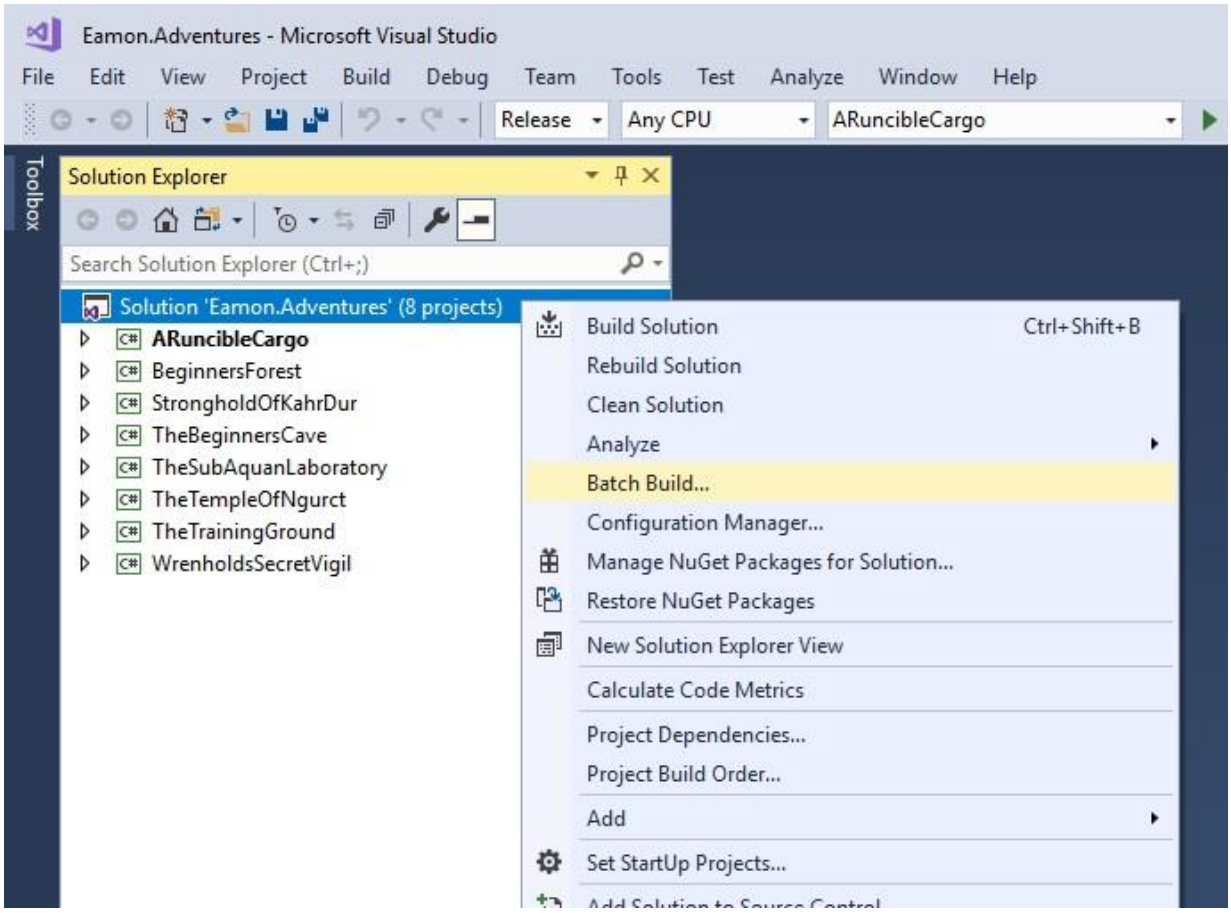
1. Make a backup of your repository directory if you downloaded a .zip file. If you Git Cloned the system, you can make a new development branch and focus your work there. Your original branch (or gameplay branch) will remain pristine.

2. In the development directory or branch run EnterMainHallUsingAdventures.bat, and send a character into the adventure you want to step through. You can save the game if you wish, but it is not required; only that the character remains in the adventure. Quit the game, but don't Quit Hall. In this example, we'll step through The Sub-Aquan Laboratory.

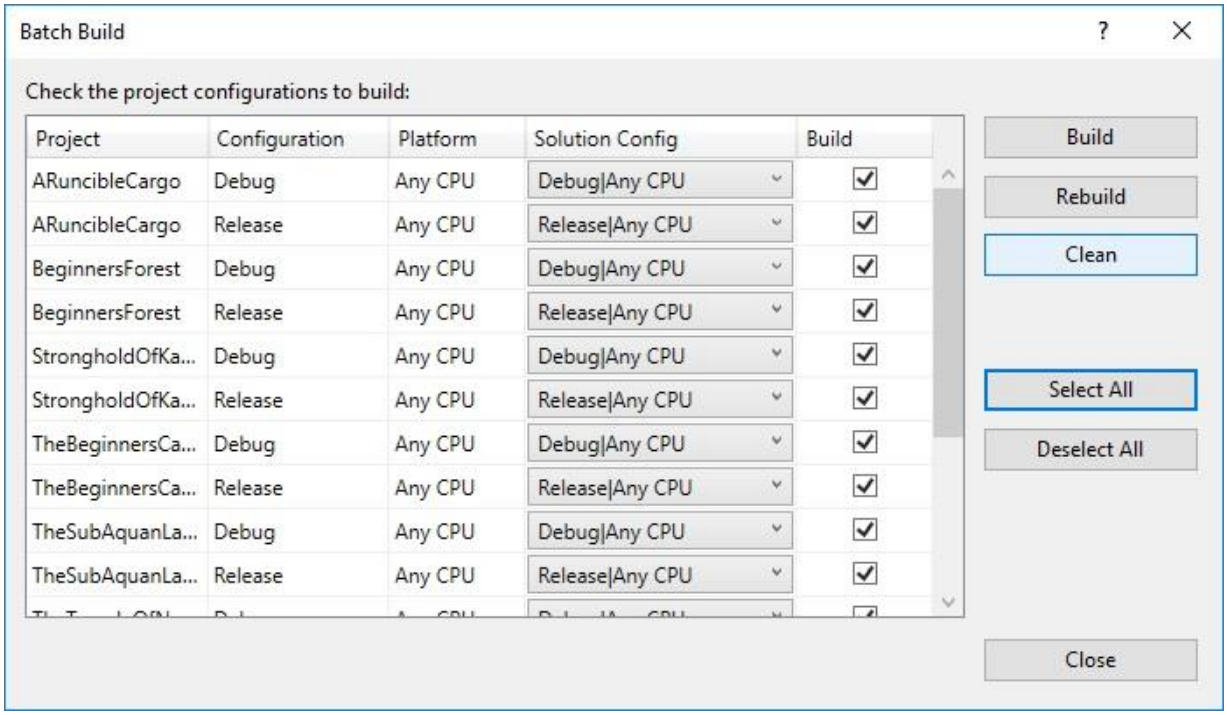
3. Open the Eamon.Adventures solution with Visual Studio.



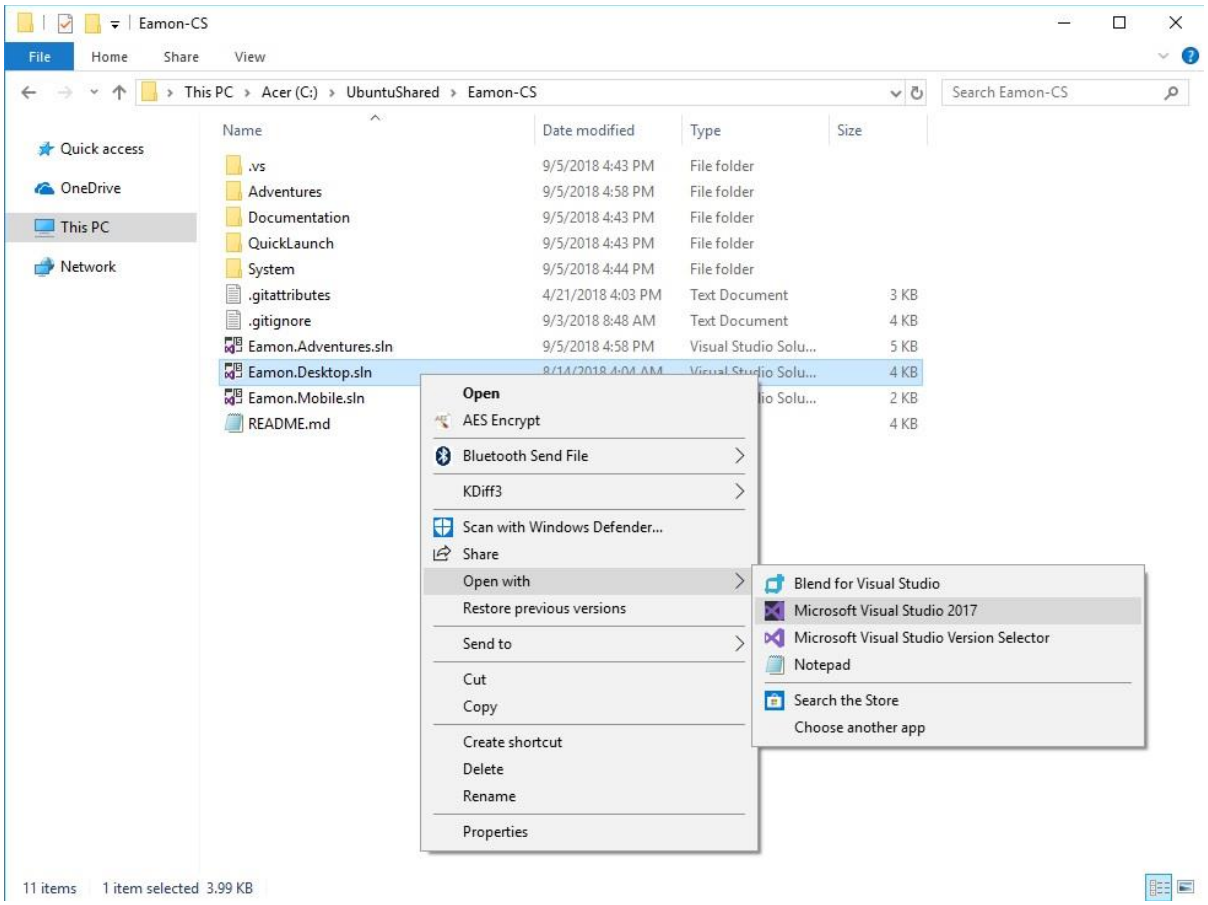
4. Once Visual Studio loads the game, right-click on the Solution node and select Batch Build.

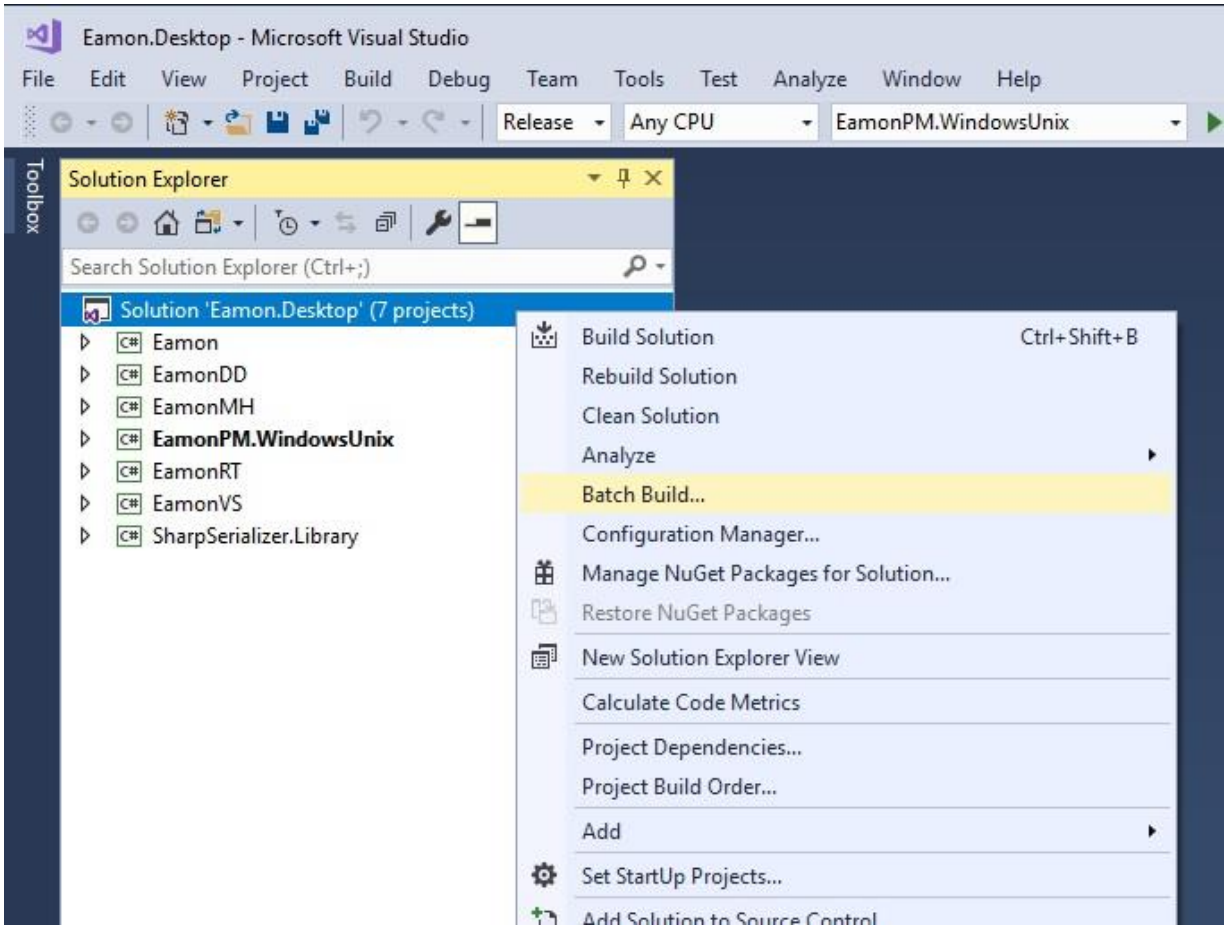


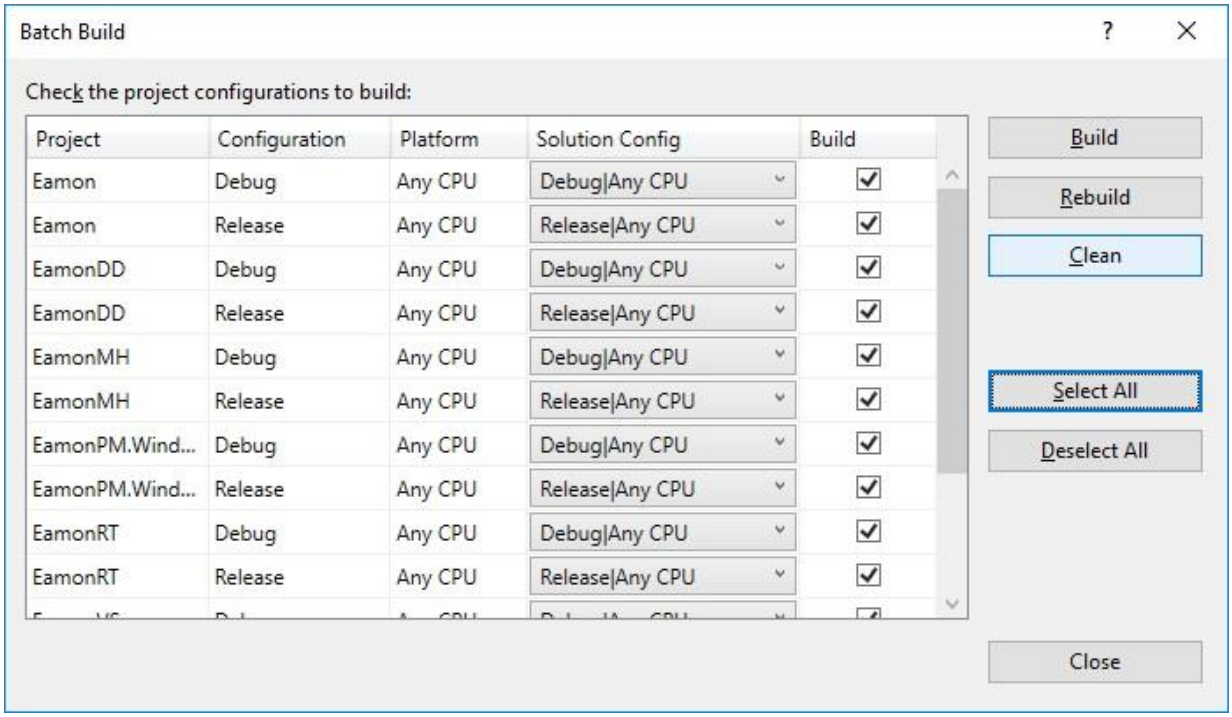
5. Click the Select All button, then click Clean.



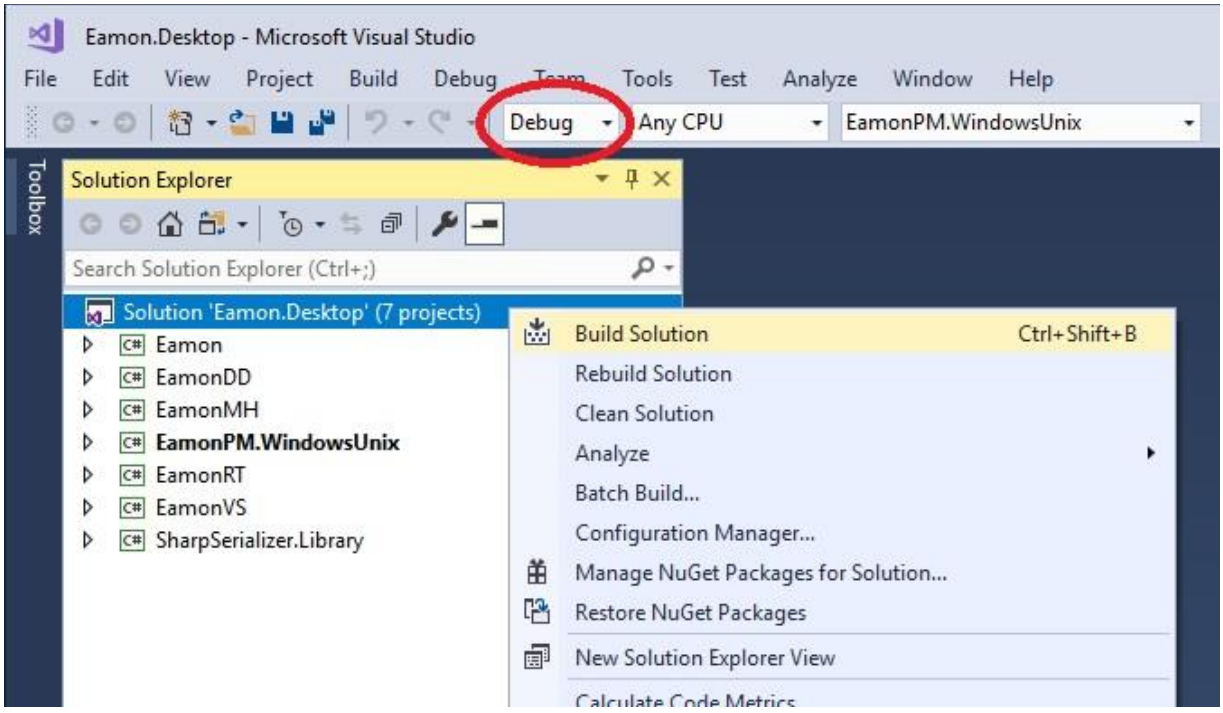
6. Repeat steps 3 through 5 using the Eamon.Desktop solution. You will probably want to open a separate copy of Visual Studio 2019 to do so since you will be recompiling both solutions shortly.



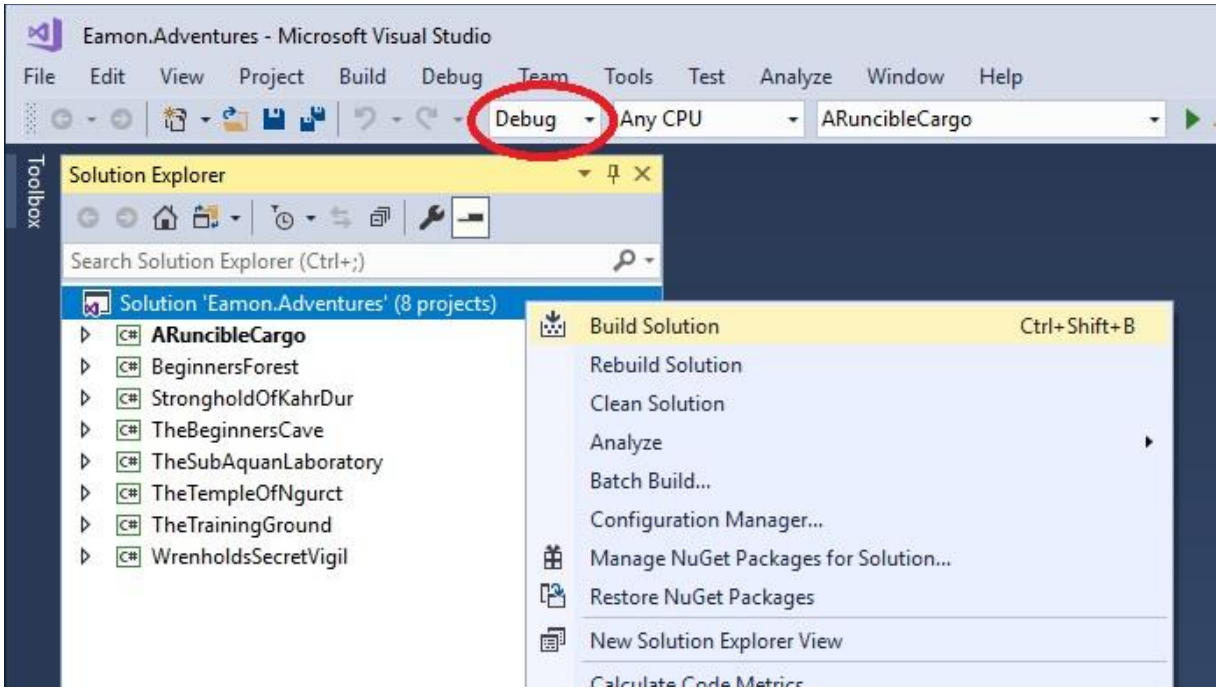

Eamon.Desktop - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Release
Any CPU
EamonPM.WindowsUnix
Toolbox
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Eamon.Desktop' (7 projects)
Eamon
EamonDD
EamonMH
EamonPM.WindowsUnix
EamonRT
EamonVS
SharpSerializer.Library
Build Solution Ctrl+Shift+B
Rebuild Solution
Clean Solution
Analyze
Batch Build...
Configuration Manager...
Manage NuGet Packages for Solution...
Restore NuGet Packages
New Solution Explorer View
Calculate Code Metrics
Project Dependencies...
Project Build Order...
Add
Set StartUp Projects...
Add Solution to Source Control


Batch Build
Check the project configurations to build:
Project
Configuration
Platform
Solution Config
Build
Eamon
Debug
Any CPU
Debug|Any CPU
Eamon
Release
Any CPU
Release|Any CPU
EamonDD
Debug
Any CPU
Debug|Any CPU
EamonDD
Release
Any CPU
Release|Any CPU
EamonMH
Debug
Any CPU
Debug|Any CPU
EamonMH
Release
Any CPU
Release|Any CPU
EamonPM.Wind...
Debug
Any CPU
Debug|Any CPU
EamonPM.Wind...
Release
Any CPU
Release|Any CPU
EamonRT
Debug
Any CPU
Debug|Any CPU
EamonRT
Release
Any CPU
Release|Any CPU
Build
Rebuild
Clean
Select All
Deselect All
Close

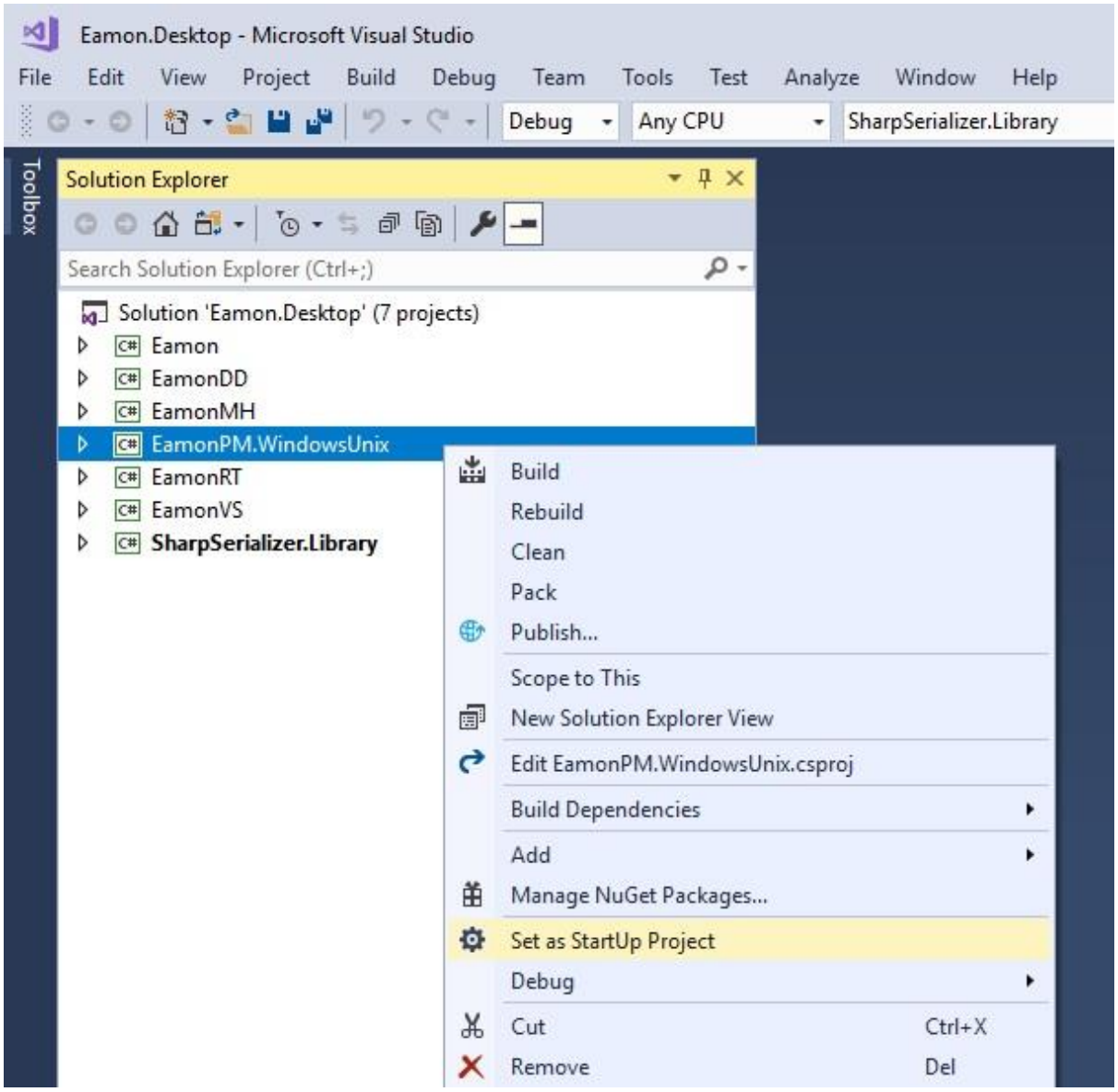
7. Ensure the system is in Debug mode (shown inside red circle). In the Eamon.Desktop solution, right-click on the Solution node and click Build Solution.



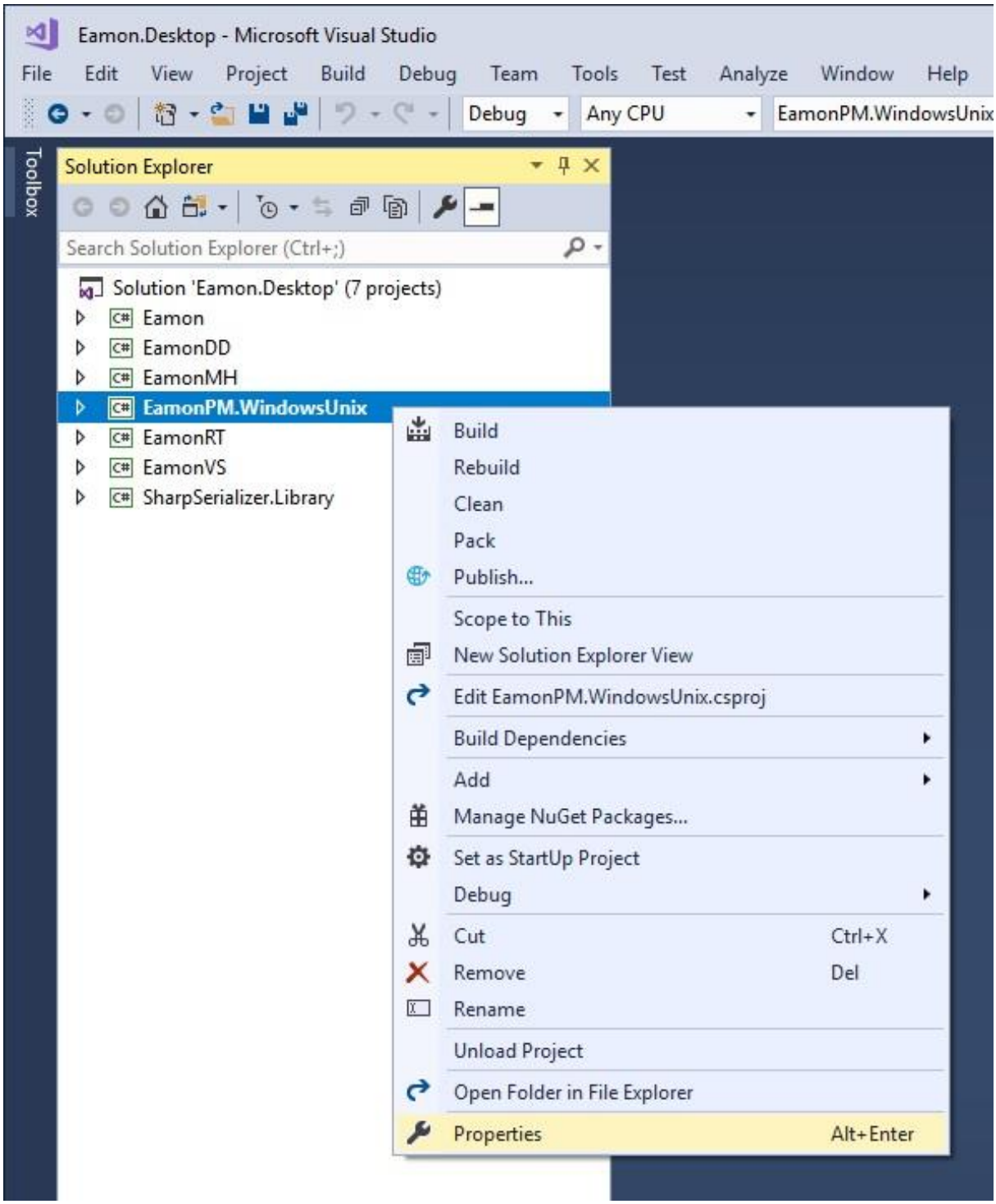
8. Ensure the system is in Debug mode (shown inside red circle). In the Eamon.Adventures solution, right-click on the Solution node and click Build Solution.



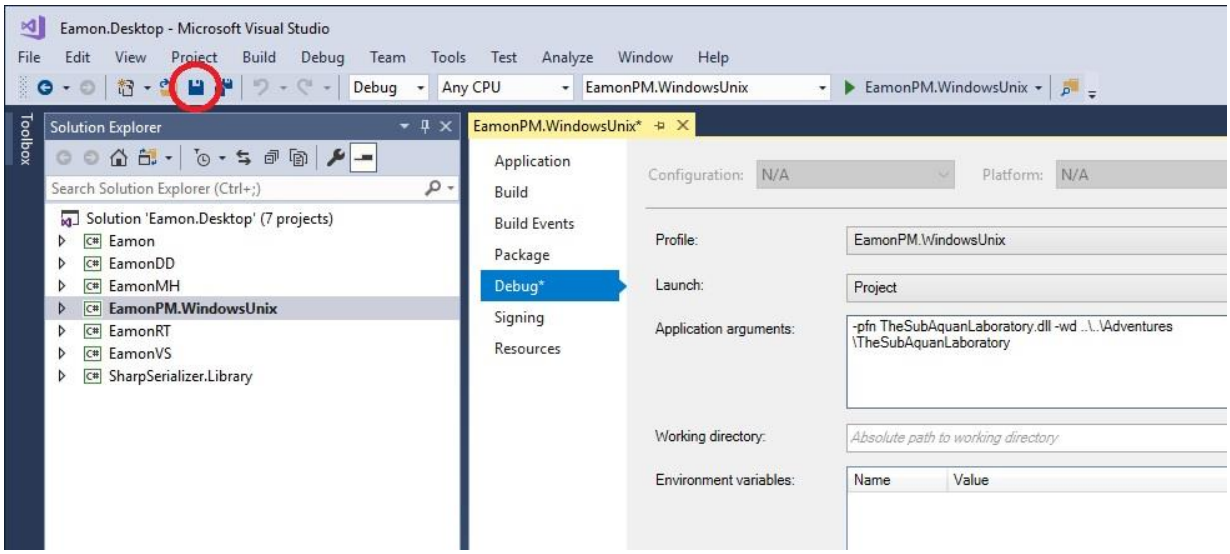
9. There should be no errors in the previous steps (see Output Windows). Right-click on the EamonPM.WindowsUnix project and select Set as Startup Project.

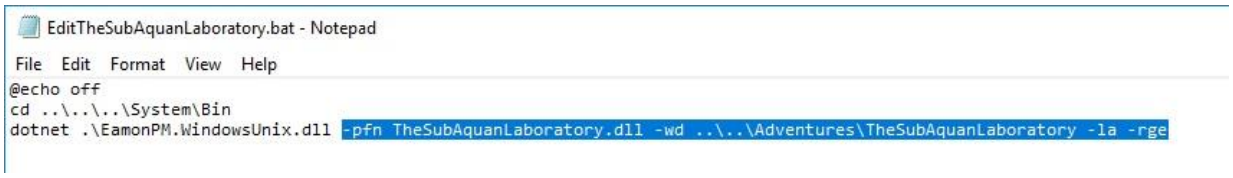


10. Right-click on the EamonPM.WindowsUnix project and select Properties.

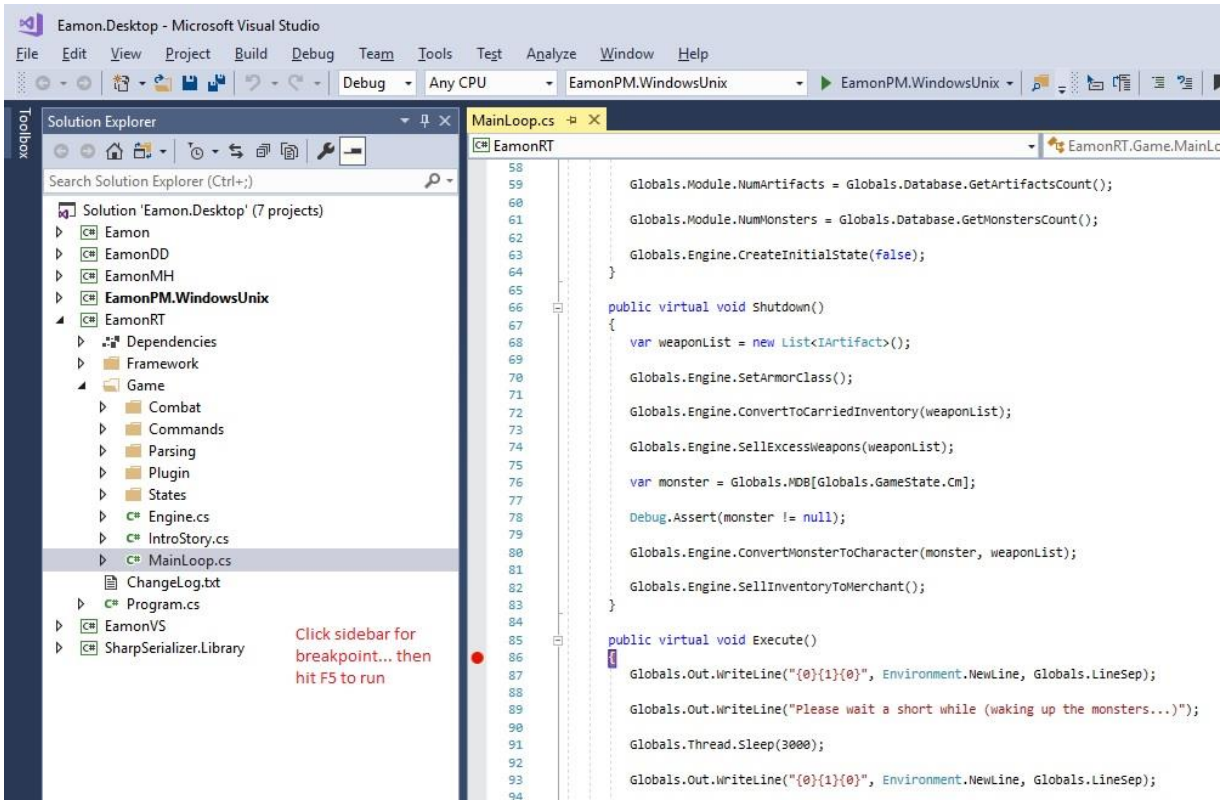


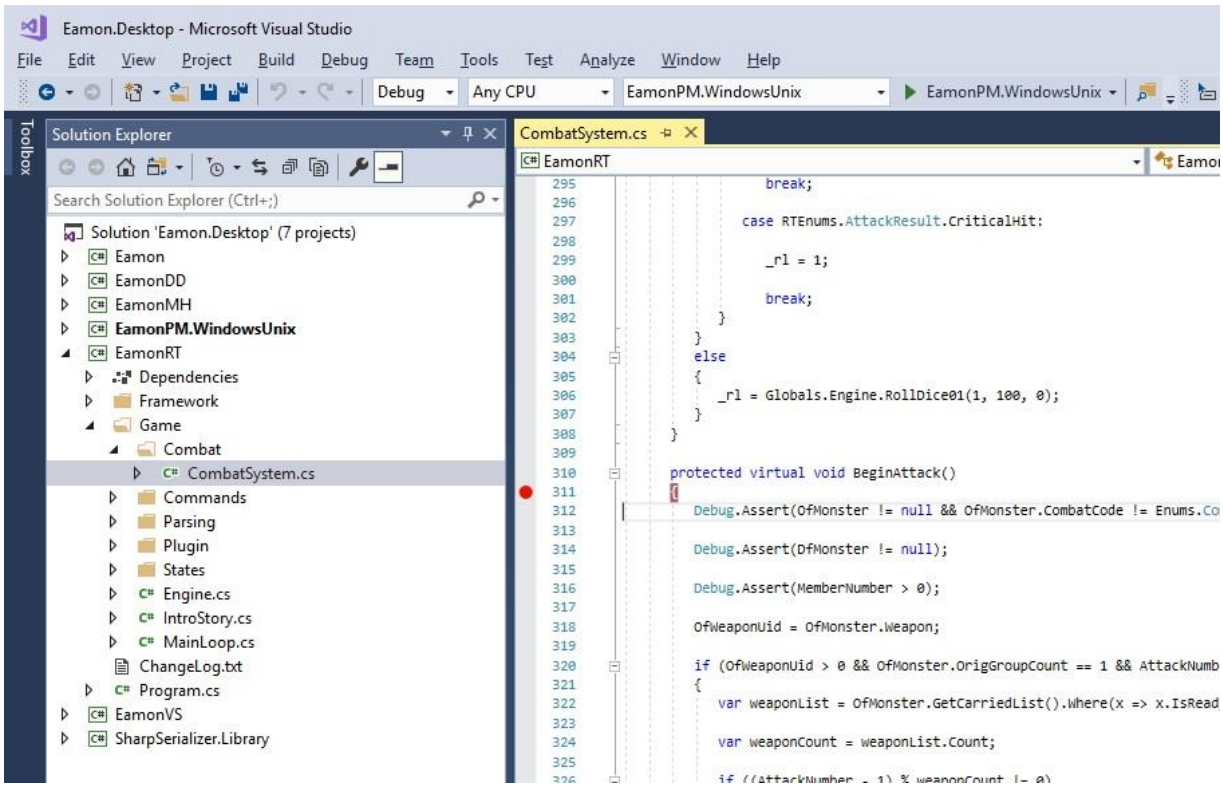
11. Set the Application arguments to the appropriate command line string. In this case, the string was taken directly from ResumeTheSubAquanLaboratory.bat. Make sure you save the changes (click the icon inside the red circle). Side note: you can use any string from any .bat file, depending on which program you want to step through. This includes the Main Hall or the Dungeon Designer. To debug these, you don't need to send a character on an adventure first. See the two examples below.

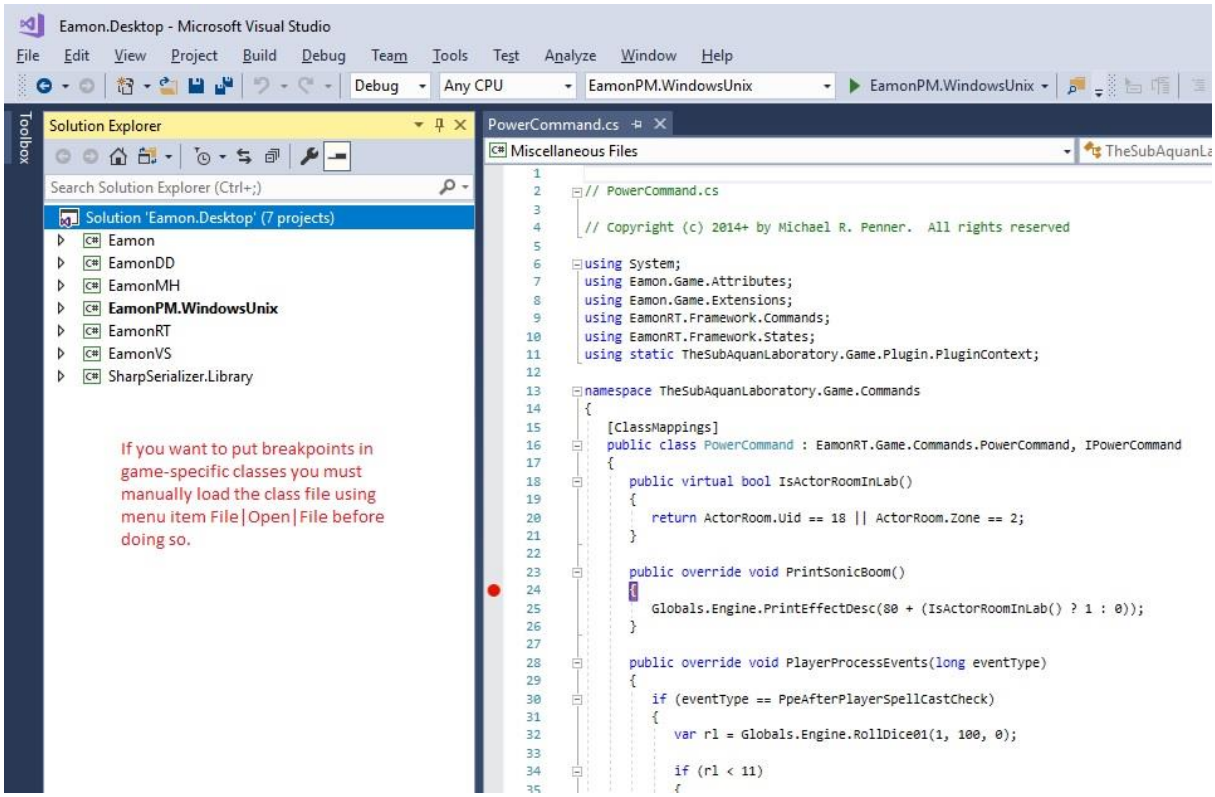




12. Put your breakpoints in the code. In the following examples, the program will break just as it enters the game's main loop, and also as an attack occurs in combat. The final example shows how you must manually load a game's source code file to put a breakpoint in it. You can set breakpoints anywhere you like in the running program; in our case here, that includes Eamon.dll, EamonRT.dll, and TheSubAquanLaboratory.dll.



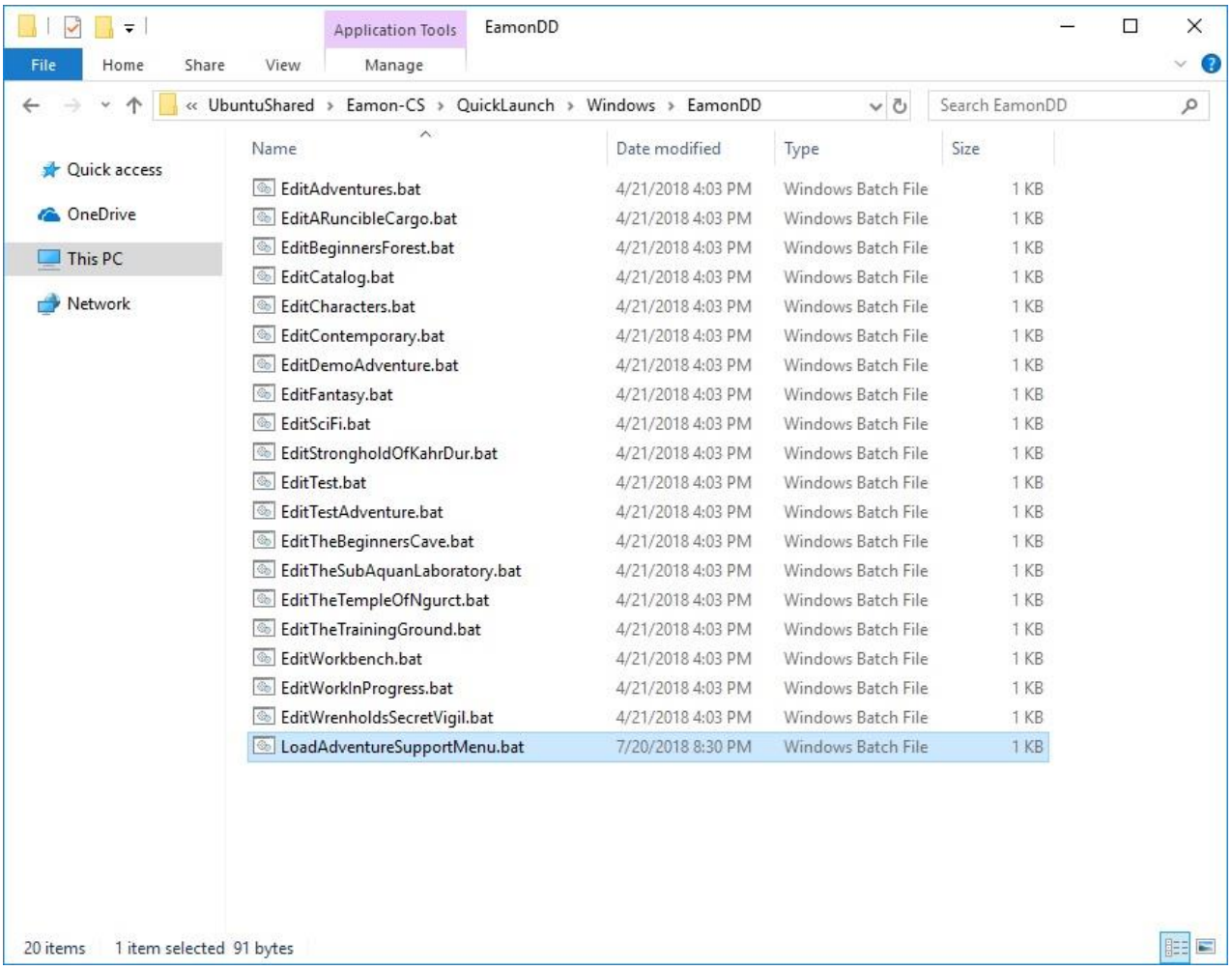

Eamon.Desktop - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
EamonPM.WindowsUnix
EamonPM.WindowsUnix
Toolbox
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Eamon.Desktop' (7 projects)
Eamon
EamonDD
EamonMH
EamonPM.WindowsUnix
EamonRT
Dependencies
Framework
Game
Combat
CombatSystem.cs
Commands
Parsing
Plugin
States
Engine.cs
IntroStory.cs
MainLoop.cs
ChangeLog.txt
Program.cs
EamonVS
SharpSerializer.Library
CombatSystem.cs
EamonRT
break;
case RTEnums.AttackResult.CriticalHit:
_rl = 1;
break;
}
}
else
{
_rl = Globals.Engine.RollDice01(1, 100, 0);
}
}
protected virtual void BeginAttack()
{
Debug.Assert(OfMonster != null && OfMonster.CombatCode != Enums.Co
Debug.Assert(DfMonster != null);
Debug.Assert(MemberNumber > 0);
OfWeaponUid = OfMonster.Weapon;
if (OfWeaponUid > 0 && OfMonster.OrigGroupCount == 1 && AttackNumb
{
var weaponList = OfMonster.GetCarriedList().Where(x => x.IsRead
var weaponCount = weaponList.Count;


Eamon.Desktop - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
EamonPM.WindowsUnix
EamonPM.WindowsUnix
Toolbox
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Eamon.Desktop' (7 projects)
Eamon
EamonDD
EamonMH
EamonPM.WindowsUnix
EamonRT
EamonVS
SharpSerializer.Library
If you want to put breakpoints in game-specific classes you must manually load the class file using menu item File|Open|File before doing so.
PowerCommand.cs
Miscellaneous Files
TheSubAquanLa
// PowerCommand.cs
// Copyright (c) 2014+ by Michael R. Penner. All rights reserved
using System;
using Eamon.Game.Attributes;
using Eamon.Game.Extensions;
using EamonRT.Framework.Commands;
using EamonRT.Framework.States;
using static TheSubAquanLaboratory.Game.Plugin.PluginContext;
namespace TheSubAquanLaboratory.Game.Commands
{
[ClassMappings]
public class PowerCommand : EamonRT.Game.Commands.PowerCommand, IPowerCommand
{
public virtual bool IsActorRoomInLab()
{
return ActorRoom.Uid == 18 || ActorRoom.Zone == 2;
}
public override void PrintSonicBoom()
{
Globals.Engine.PrintEffectDesc(80 + (IsActorRoomInLab() ? 1 : 0));
}
public override void PlayerProcessEvents(long eventType)
{
if (eventType == PpeAfterPlayerSpellCastCheck)
{
var rl = Globals.Engine.RollDice01(1, 100, 0);
if (rl < 11)

13. Press F5 to run the program. The game will launch just as if you had double-clicked the ResumeTheSubAquanLaboratory.bat file. If you saved any games, you can restore them at this point as you typically would. You can also start the program or step through it line by line at any point using F10 (step over) or F11 (step into).

BUILDING NEW ADVENTURES

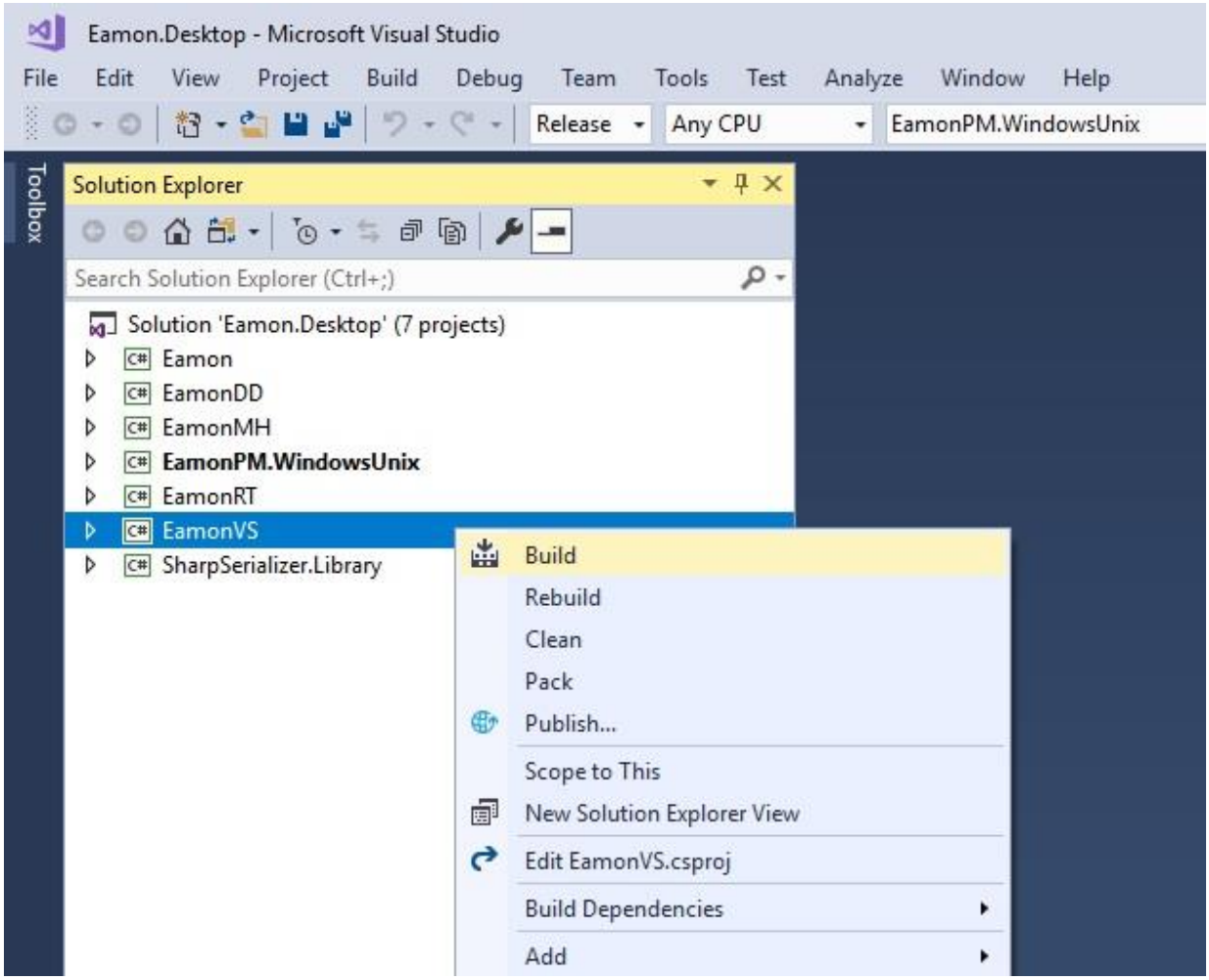
With Eamon CS, it is still possible to send adventurers to their death for fun and profit, even after all these years! You can create new games with the tools available to you. Standard adventures - that is, those that use the EamonRT generic game engine and have no custom programming - can be built without Visual Studio 2019 Community Edition installed. But Custom adventures (like The Beginner's Cave, etc.) require the VS tool suite to be on your system. Up to this point, it has been a manual effort to bootstrap new games. But significant progress has been made towards fully automating the process. You will find in the EamonDD folder under QuickLaunch a new .bat or .sh file that when run, will enable an Adventure Support Menu under Utilities. This file is LoadAdventureSupportMenu:



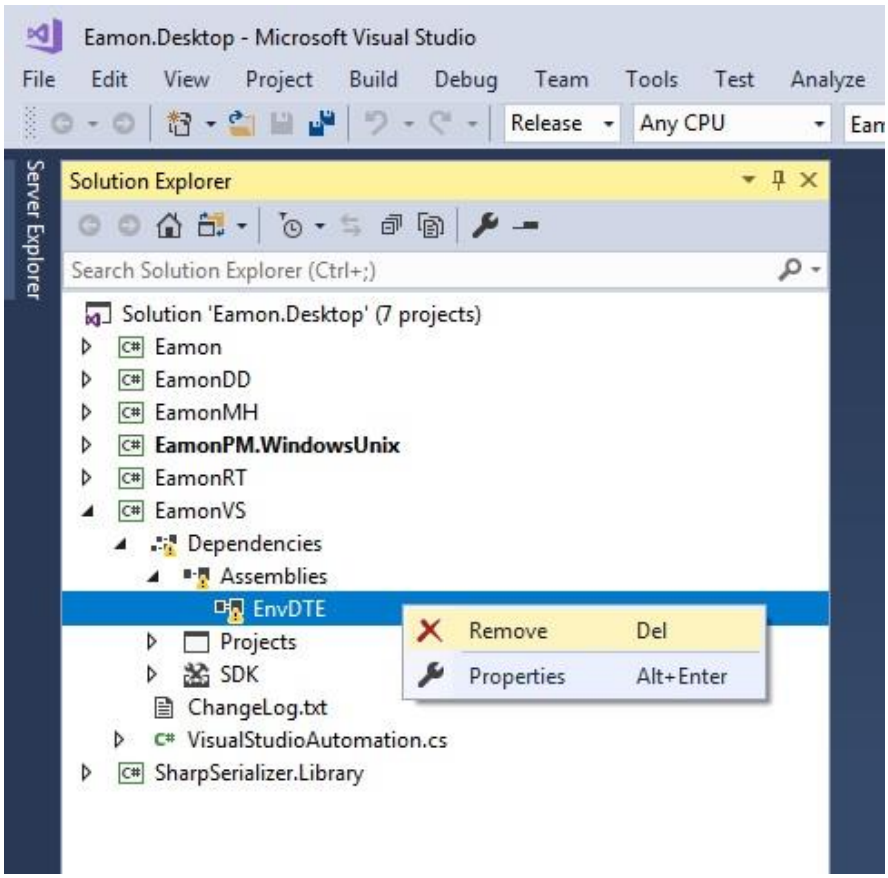
Depending on your system configuration, there may be a few manual steps that you must do if you plan to create Custom adventures. Perform these steps if necessary, before launching this menu for the first time.

1. There is a new project in Eamon.Desktop called EamonVS that integrates with the Microsoft EnvDTE automation library. EnvDTE is omitted from Eamon CS as it is not on Microsoft's redistribution list. But it was placed on your system when you installed Visual Studio. EamonVS includes it as a reference. The EamonDD program uses EamonVS to automate the process of adding new Custom adventure projects to Eamon.Adventures and to recompile the solution. Note the following two scenarios:

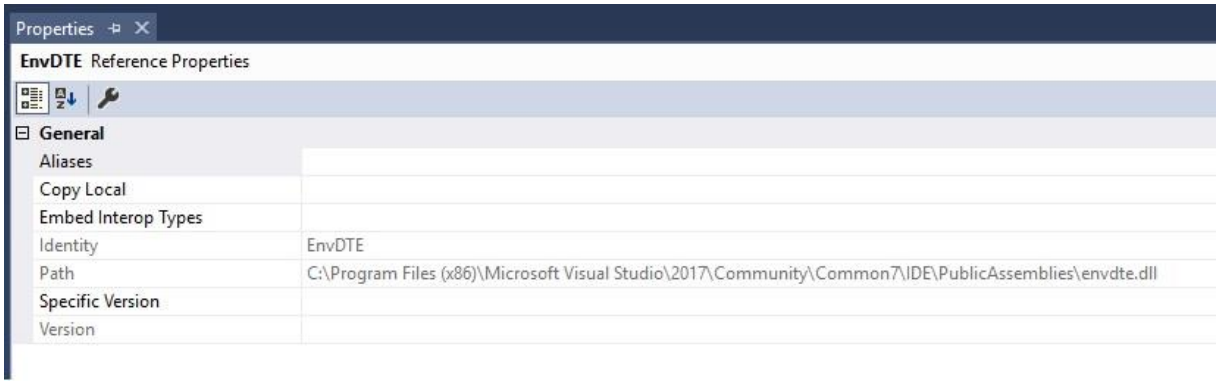
2. Scenario #1: If you are using Visual Studio 2019 Community, installed in the default location, you should be ready to go right now. Open Eamon.Desktop, right-click on the EamonVS project and Build it. If the Build succeeds, skip to page 42 of this document. (If the Build fails because it can't find EnvDTE.dll, you're in Scenario #2 - see below).



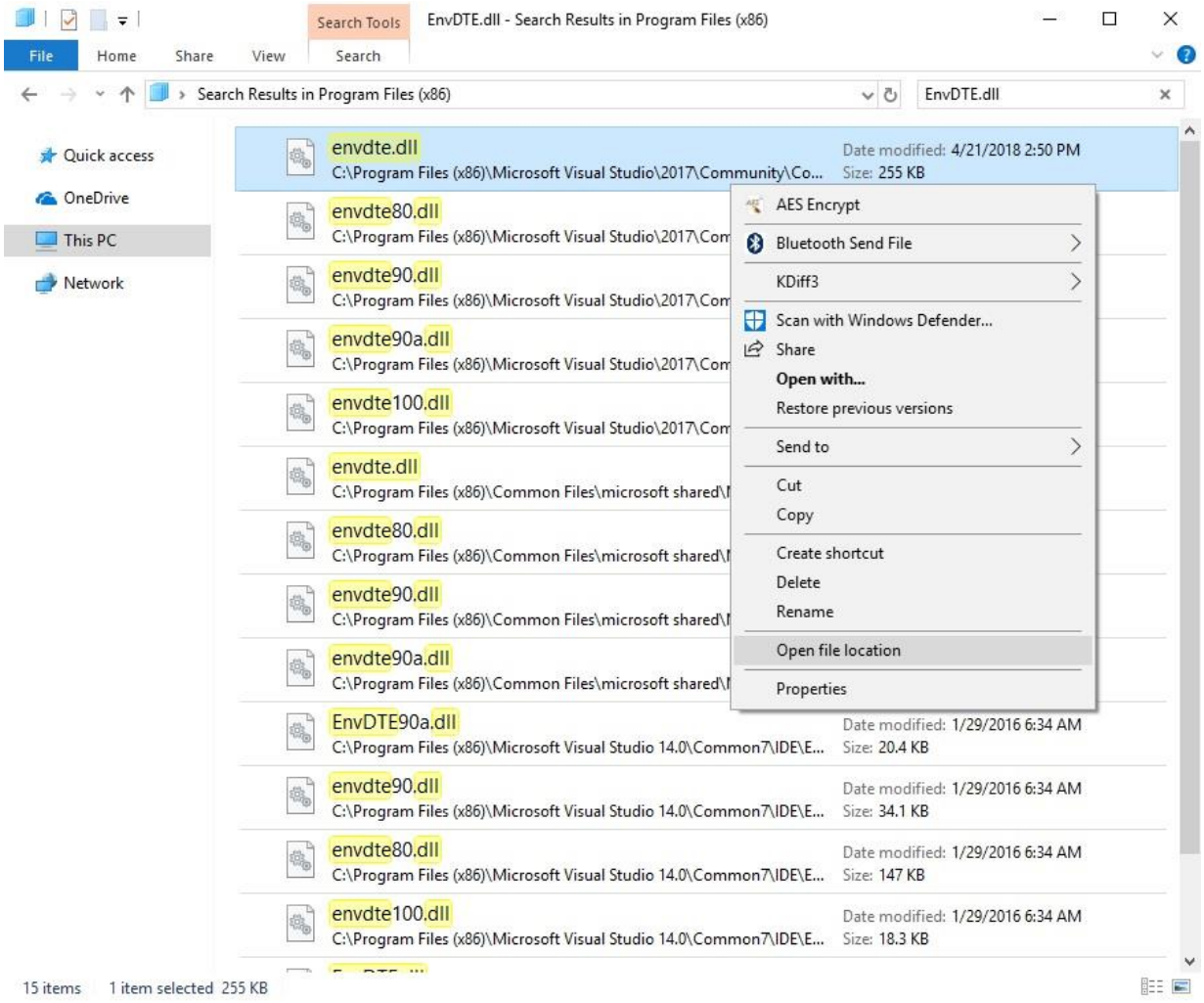
3. Scenario #2: If you are using Visual Studio 2019, installed in a non-default location, or you're using a non-Community Edition (e.g., Professional or Enterprise). You will have to remove the EnvDTE reference from EamonVS, locate EnvDTE in your Visual Studio directory hierarchy, and re-add it as a reference to EamonVS. The EnvDTE library should be about 256KB in size. Once you do this, you should be able to compile EamonVS.



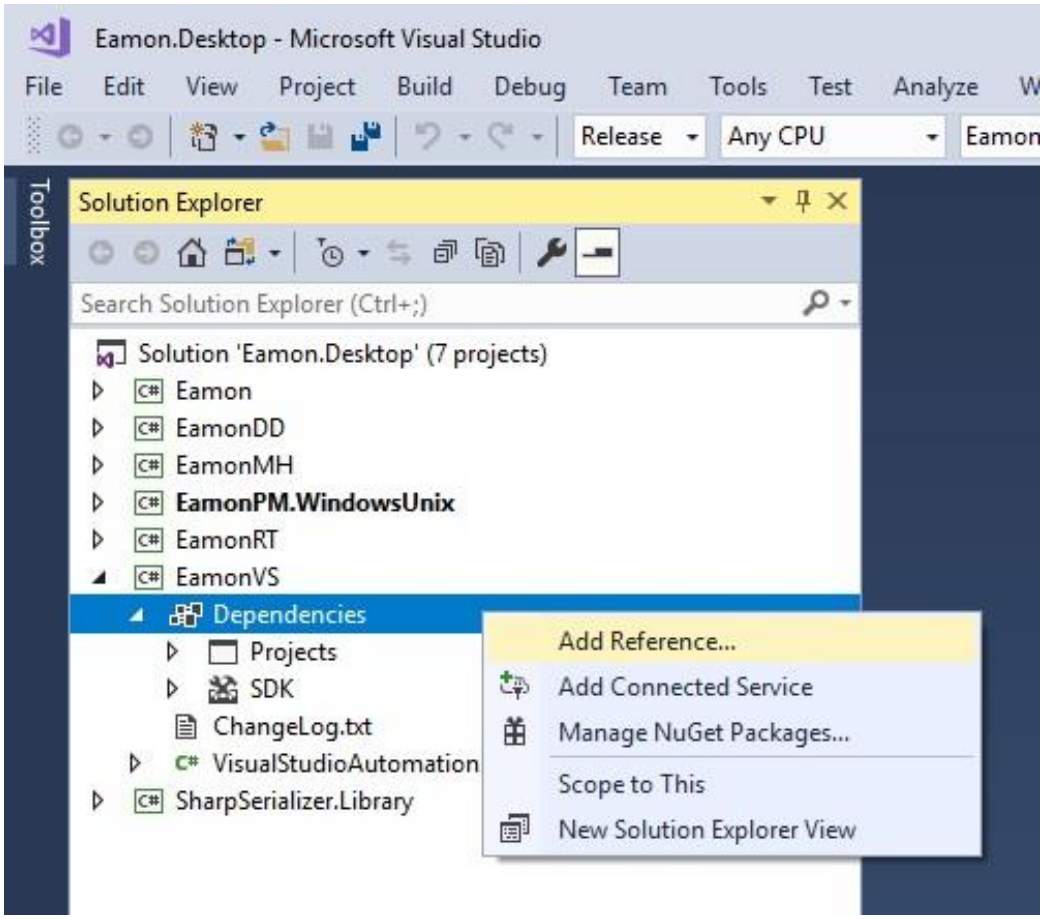
4. Note the path originally supplied for EnvDTE.dll; this will give you an idea on where to look for this library on your system.



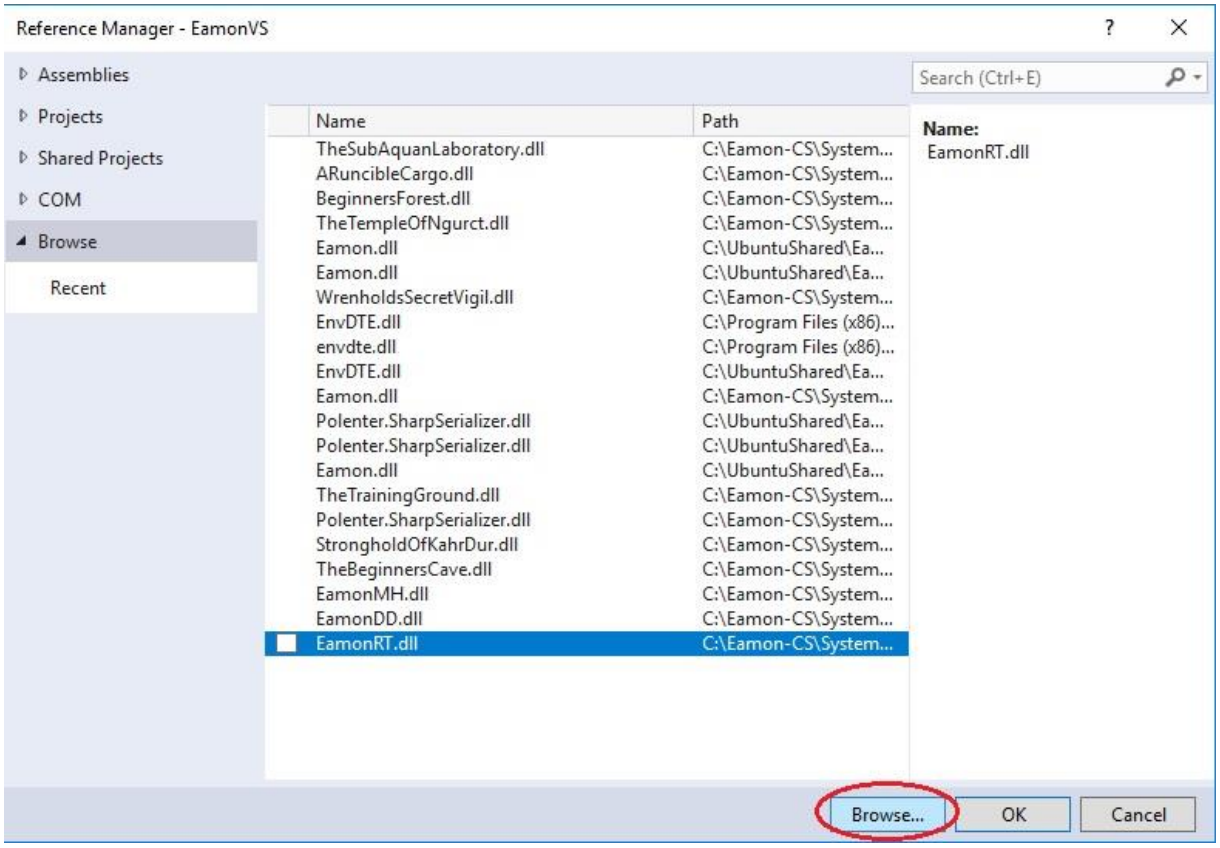
5. The easiest way to locate EnvDTE.dll is to do a file search in the directory hierarchy above your Visual Studio install. For example, here, the search is done in Program Files (x86). Then right-click on the correct file and choose Open File Location. Copy the full directory path for use in the following steps.



6. Right-click on Dependencies for the EamonVS project and choose Add Reference.



7. Choose Browse to bring up a file browser. You can locate EnvDTE.dll using the path discovered in step #5 and add it as a dependency to EamonVS. Then compile EamonVS as noted in step #2.



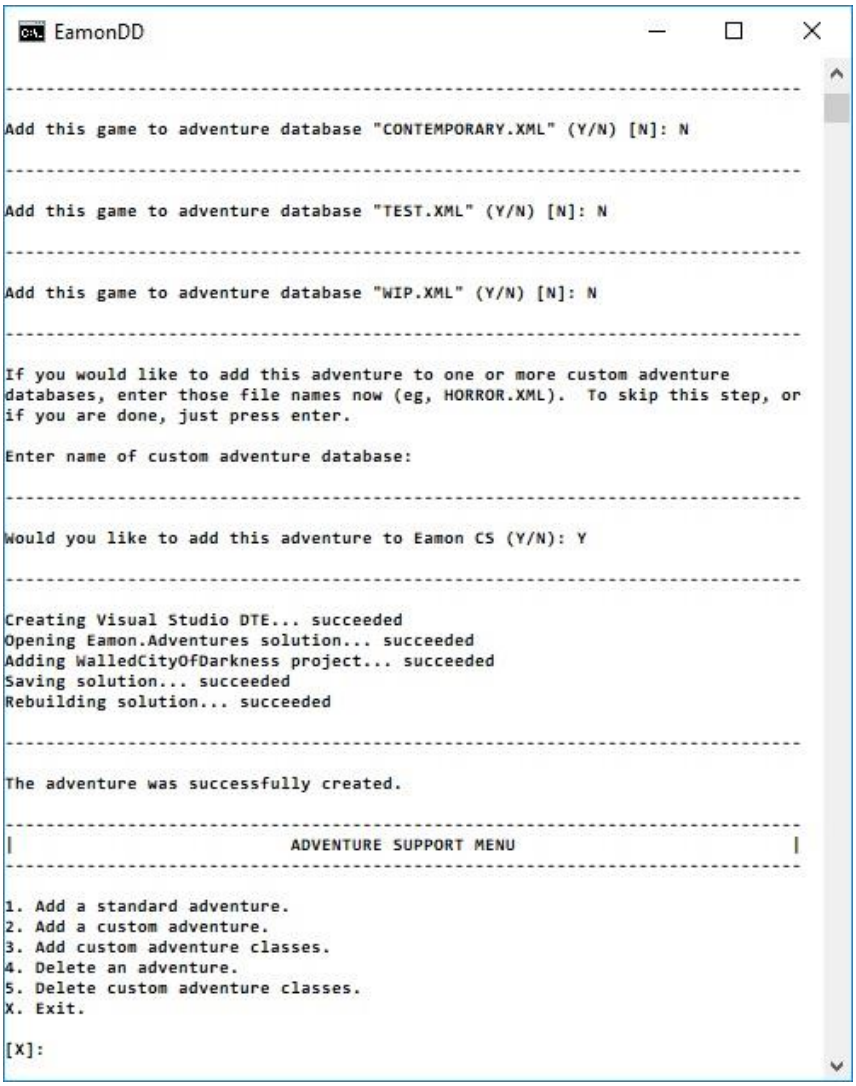
8. There is one additional step you must do for Scenario #2. You need to locate the devenv.exe program, which should be in the IDE directory, immediately above EnvDTE.dll. Edit the LoadAdventureSupportMenu.bat file (but not .sh) and append to the end of the dotnet launch string the -dep command line flag along with the full path and file name:

Before proceeding further, if you downloaded the .zip file for Eamon CS, it might make sense to back up your repository so you can revert if necessary. For those who cloned the repository from GitHub, it might make sense to create a branch for your new adventure so you can back out at any time.

Once you have performed any manual steps listed above, you will be able to run LoadAdventureSupportMenu. Creating adventures requires entering just a few key pieces of data. The automated process uses resources embedded within the EamonDD plugin:

- Creates the game folder under Adventures
- Generates all needed code (Custom adventures only)
- Creates all necessary QuickLaunch .bat and .sh files
- Adds game to appropriate Adventure database(s)
- Adds game project to Eamon.Adventures.sln (Custom adventures only)
- Rebuilds Eamon.Adventures.sln, producing game library (Custom adventures only)

Below we see an example of Custom adventure creation where a new game called Walled City of Darkness (Eamon #150) is bootstrapped using the Adventure Support Menu.



The only thing you need to do to play your game after all this is run the game's Edit[YourAdventureName] .bat or .sh file and add:

- A Module record
- A Room record

For Custom adventures, a complete set of derived foundational classes (e.g., Artifact, Monster, Room, Engine, etc.) are generated and available for you to modify. The generated classes should be sufficient for many Custom games. But remember you can always add your own new or overridden classes if you need to get exotic. There are two additional menu options, detailed below, that allow you to add and remove classes programmatically. Every facet of the system is built to override, so the sky is the limit here. The actual process of customization using Visual Studio 2019 is beyond the scope of this section. It will be fully documented elsewhere, although there are games available to look at for ideas.

Finally, a word about submissions: anyone who wishes to build a game and have it included in Eamon CS should contact me to discuss the matter before beginning any work. The contact info is in AUTHORS.txt under Documentation. You should always build a new game in a unique branch off master, so it cleanly merges when completed. There may be suggestions for improvements when reviewing the code in the branch. It is no cause for concern, just part of the process of building the best game possible. The goal of doing this is to enhance the experience for people who discover Eamon CS and play it in the future. At this point, I have yet to fully work out the logistics of game developer pull requests against master.

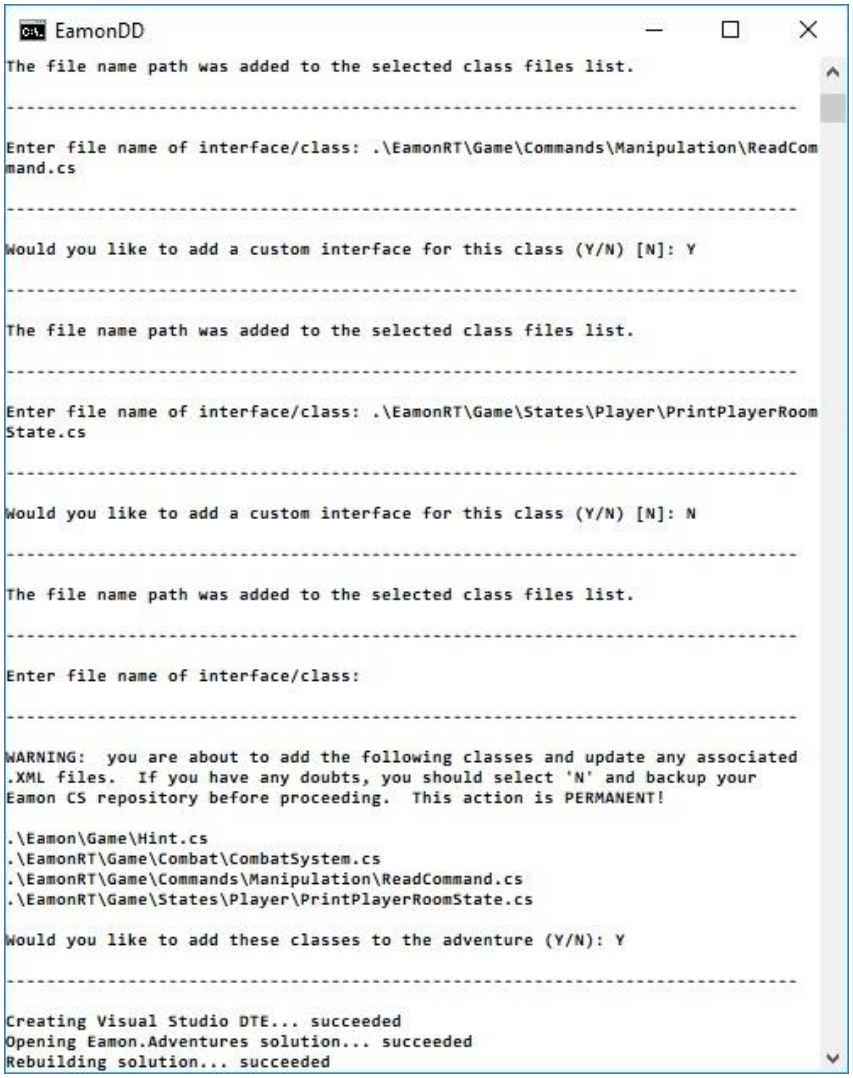
ADDING CUSTOM ADVENTURE CLASSES

Custom games are created in Eamon CS using the C# mechanism of sub-classing. The system has no sealed classes, and every property and method is declared virtual. You can override any aspect of the game engine to suit your needs. If you want to produce a complex, interesting game with lots of special effects, you will almost certainly need to create new classes for your adventure. There are numerous examples of this in the adventures contained in the Eamon.Adventures solution. It turns out, the layout of folders and namespaces in Eamon CS lends itself to a simple code generation mechanism based on a template discovered while studying the already-created games. You can use the Adventure Support Menu to generate custom classes, complete with matching interfaces (if needed), based on many classes in the Eamon, EamonDD or EamonRT libraries. These generated classes appear in the selected adventure, and then you are free to add your custom code. For any class with a corresponding .XML file (e.g., Room.cs and ROOMS.XML), the .XML file will be updated to reflect the newly added class.

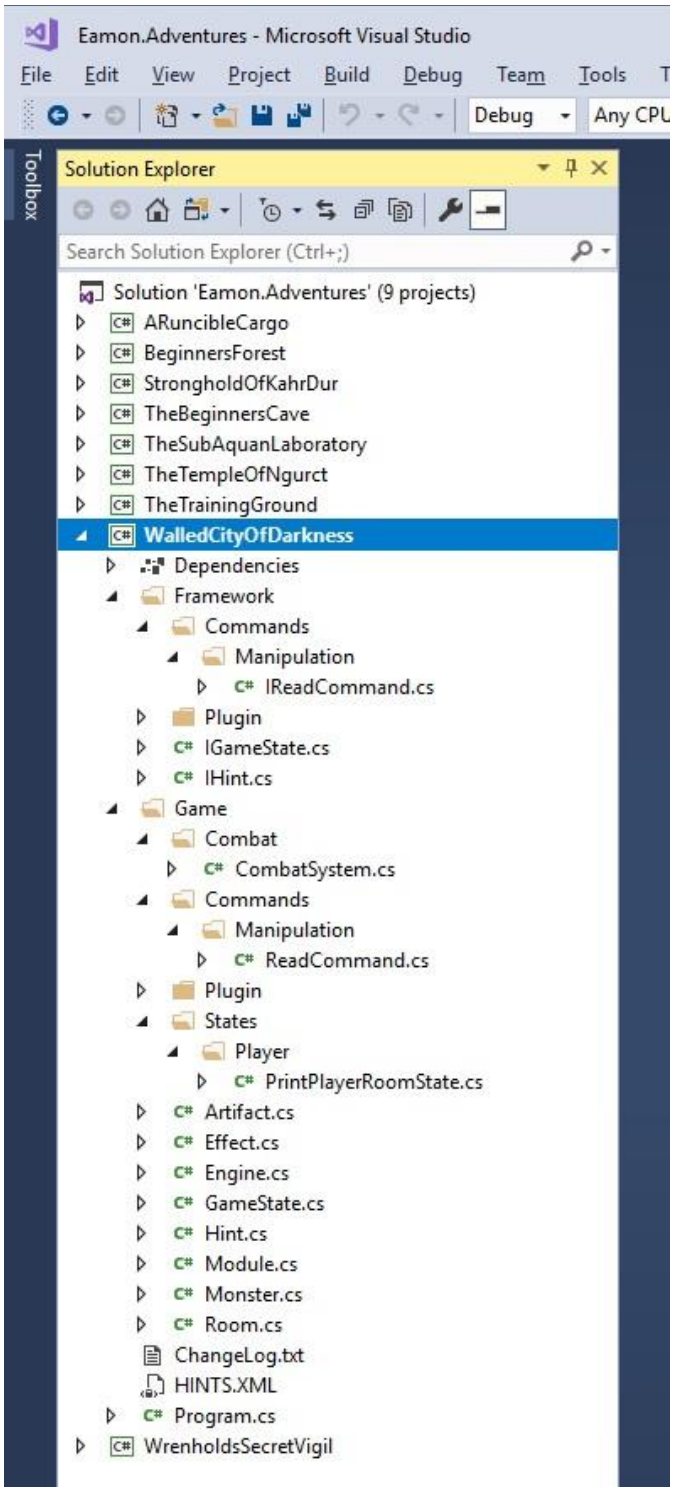
While the code generator works great with classes that fit the template, it may fail with others that don't, so this comes with a caveat. *Always verify the generated code is what you are expecting. Depending on the class, sometimes the code will be flawed, and your game won't compile; other times it will compile but be subtly wrong*. Visual Studio's Intellisense can help identify problematic code that needs fixing. Since this menu option is intended only as a bootstrapper, you can take the output verbatim or modify it to better suit your needs. Also, remember you can always bypass this convenience and craft your code by hand.

The following example demonstrates the adding of several new classes to the previously created Walled City of Darkness adventure. (Note the adventure has been pre-processed to remove the Hint.cs file.)

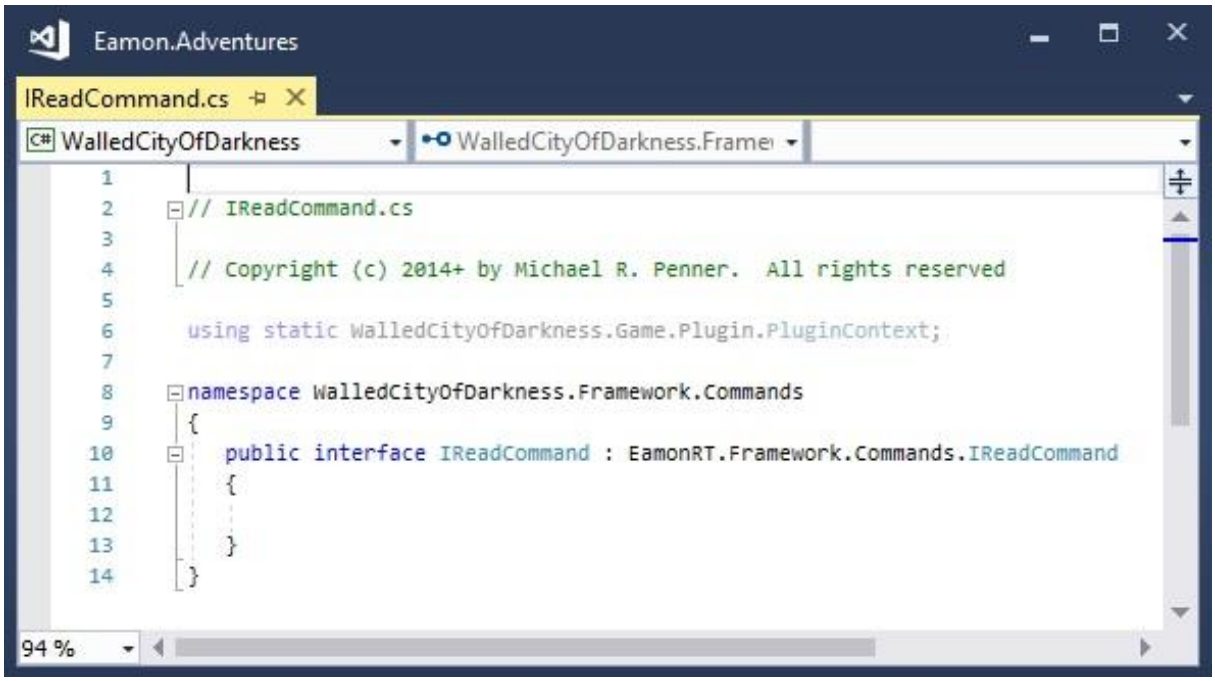
1. You have several options when using this menu item. You can generate stand-alone classes, stand-alone interfaces, or class/interface pairs. If you create an independent class and then later decide to generate its matching interface, you will have to update the class manually to derive from the new interface. If you create a matching class/interface pair, the interface won't appear if it already exists. File paths are specified relative to the working directory System.

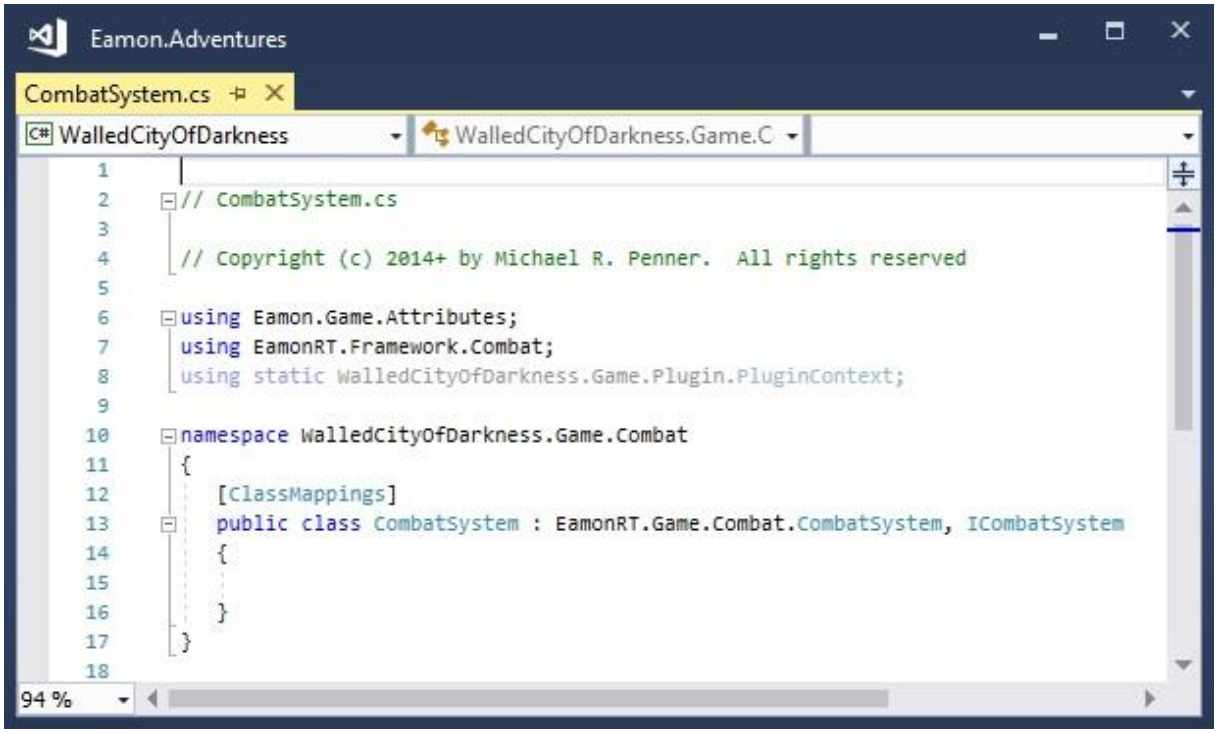


2. The code generator places classes and interfaces in the adventure folder. The system automatically builds a directory hierarchy that mirrors that of the Eamon CS base libraries. Classes go under the Game folder, interfaces under Framework. At this point, you can add your custom code.

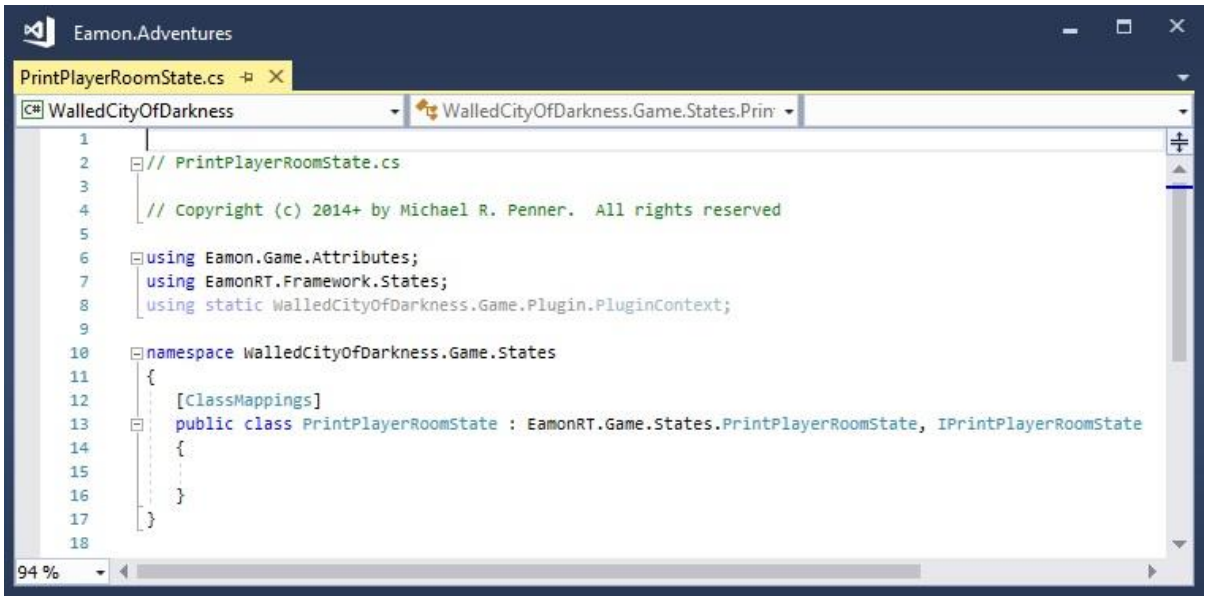


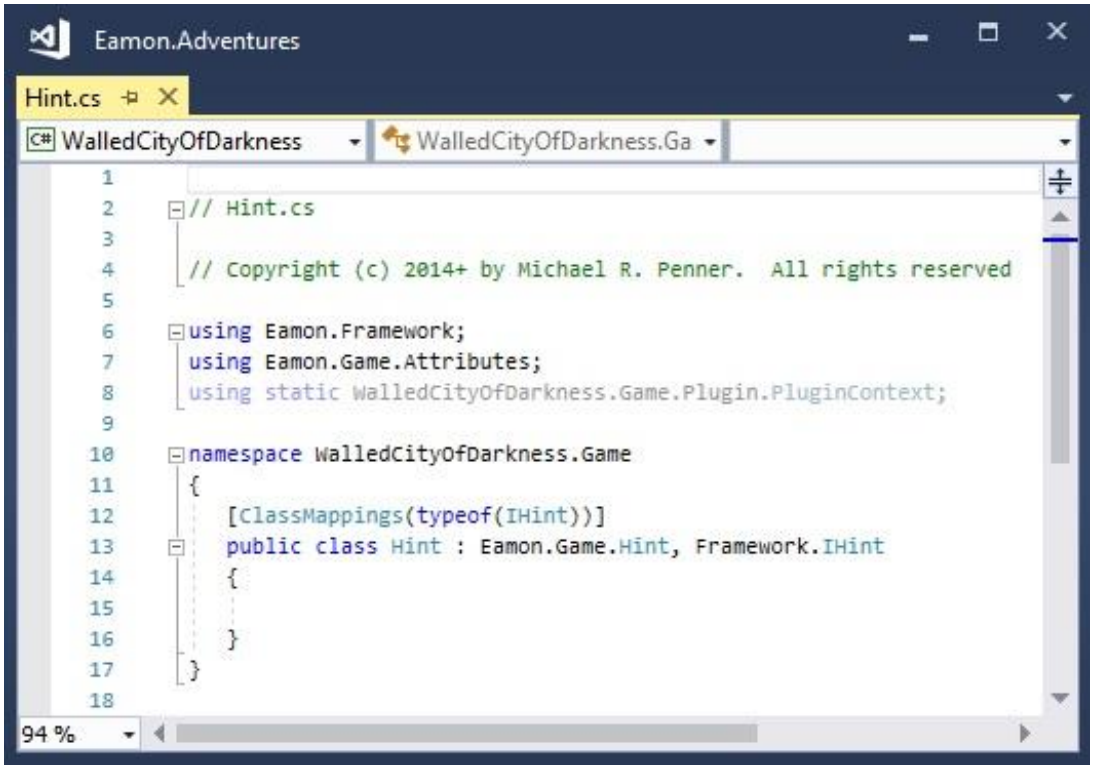
3. The following screenshots show what the code generator produced. If you compare this output with the games in Eamon.Adventures you will quickly see parallels between them. You should fix anything Intellisense flags as being invalid, and also hover over the various parts of the code to ensure the specified classes/interfaces are correct.



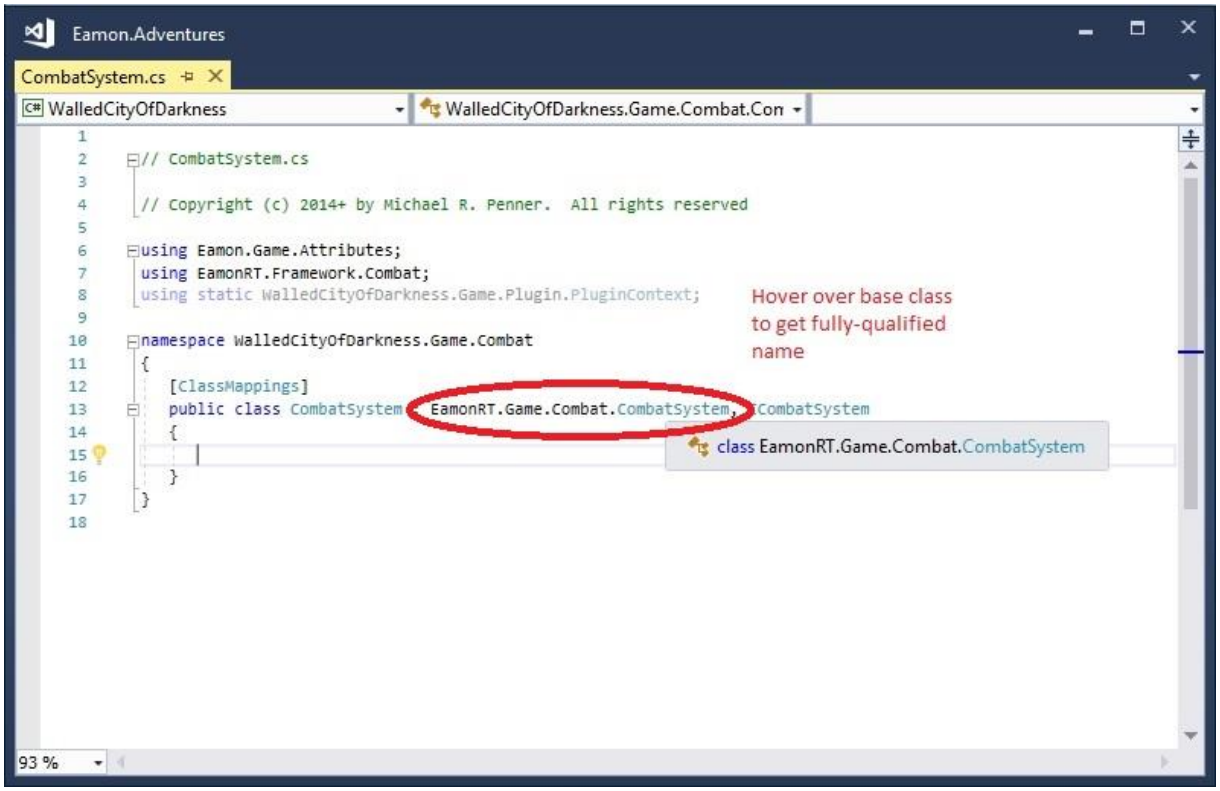


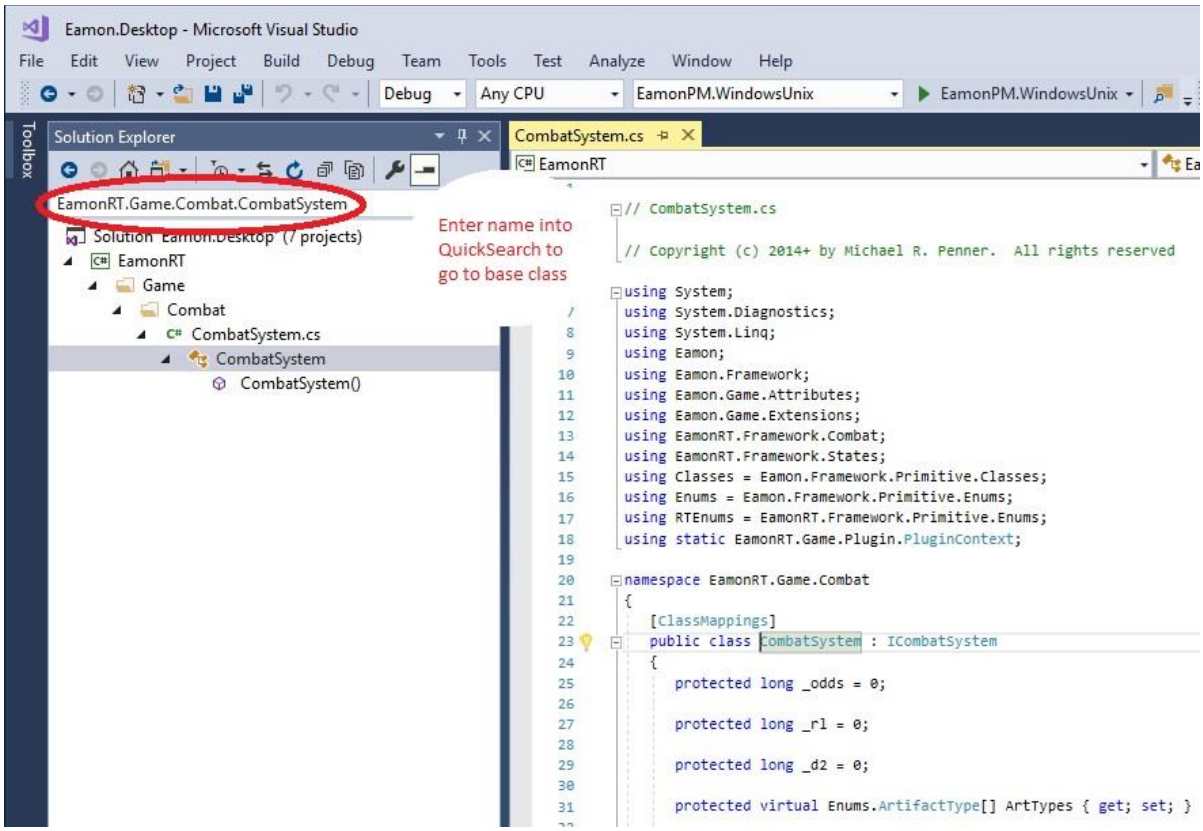

Eamon.Adventures
ReadCommand.cs
WalledCityOfDarkness
WalledCityOfDarkness.Game.Comma
// ReadCommand.cs
// Copyright (c) 2014+ by Michael R. Penner. All rights reserved
using Eamon.Game.Attributes;
using EamonRT.Framework.Commands;
using static WalledCityOfDarkness.Game.Plugin.PluginContext;
namespace WalledCityOfDarkness.Game.Commands
{
[ClassMappings(typeof(IReadCommand))]
public class ReadCommand : EamonRT.Game.Commands.ReadCommand, Framework.Commands.IReadCommand
{
}
}
94 %


Eamon.Adventures
PrintPlayerRoomState.cs
WalledCityOfDarkness
WalledCityOfDarkness.Game.States.Prin
// PrintPlayerRoomState.cs
// Copyright (c) 2014+ by Michael R. Penner. All rights reserved
using Eamon.Game.Attributes;
using EamonRT.Framework.States;
using static WalledCityOfDarkness.Game.Plugin.PluginContext;
namespace WalledCityOfDarkness.Game.States
{
[ClassMappings]
public class PrintPlayerRoomState : EamonRT.Game.States.PrintPlayerRoomState, IPrintPlayerRoomState
{
}
}
94 %


Eamon.Adventures
Hint.cs
WalledCityOfDarkness
WalledCityOfDarkness.Ga
1
2 // Hint.cs
3
4 // Copyright (c) 2014+ by Michael R. Penner. All rights reserved
5
6 using Eamon.Framework;
7 using Eamon.Game.Attributes;
8 using static WalledCityOfDarkness.Game.Plugin.PluginContext;
9
10 namespace WalledCityOfDarkness.Game
11 {
12 [ClassMappings(typeof(IHint))]
13 public class Hint : Eamon.Game.Hint, Framework.IHint
14 {
15
16 }
17 }
18
94 %

4. When adding custom code, type public override or protected override and choose the right property or method from the list presented by Intellisense. It is often useful to refer to (or use code from) the parent classes in the Eamon CS base libraries, but when the existing games migrated to the Eamon.Adventures solution they were changed to use file referencing rather than project referencing. Unfortunately, this prevents quick navigation to the parent classes using the typical right-click and Go to Definition or Go to Implementation. Doing this brings up only metadata. There are third-party tools that can overcome this limitation using library de-compilation, but in general, they are commercial products with no Community Edition available. To overcome this, you can open two copies of Visual Studio, one with Eamon.Desktop loaded, the other with Eamon.Adventures. Then when you want to access a parent class, method or property, type the name into the Search Bar and double click the found item. Doing this appears to be the best option available.




Eamon.Desktop - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
EamonPM.WindowsUnix
Toolbox
Solution Explorer
EamonRT.Game.Combat.CombatSystem
Solution 'Eamon.Desktop' (7 projects)
EamonRT
Game
Combat
CombatSystem.cs
CombatSystem
CombatSystem()
Enter name into QuickSearch to go to base class
CombatSystem.cs
EamonRT
// CombatSystem.cs
// Copyright (c) 2014+ by Michael R. Penner. All rights reserved
using System;
using System.Diagnostics;
using System.Linq;
using Eamon;
using Eamon.Framework;
using Eamon.Game.Attributes;
using Eamon.Game.Extensions;
using EamonRT.Framework.Combat;
using EamonRT.Framework.States;
using Classes = Eamon.Framework.Primitive.Classes;
using Enums = Eamon.Framework.Primitive.Enums;
using RTEnums = EamonRT.Framework.Primitive.Enums;
using static EamonRT.Game.Plugin.PluginContext;
namespace EamonRT.Game.Combat
{
[ClassMappings]
public class CombatSystem : ICombatSystem
{
protected long _odds = 0;
protected long _rl = 0;
protected long _d2 = 0;
protected virtual Enums.ArtifactType[] ArtTypes { get; set; }

DELETING CUSTOM ADVENTURE CLASSES

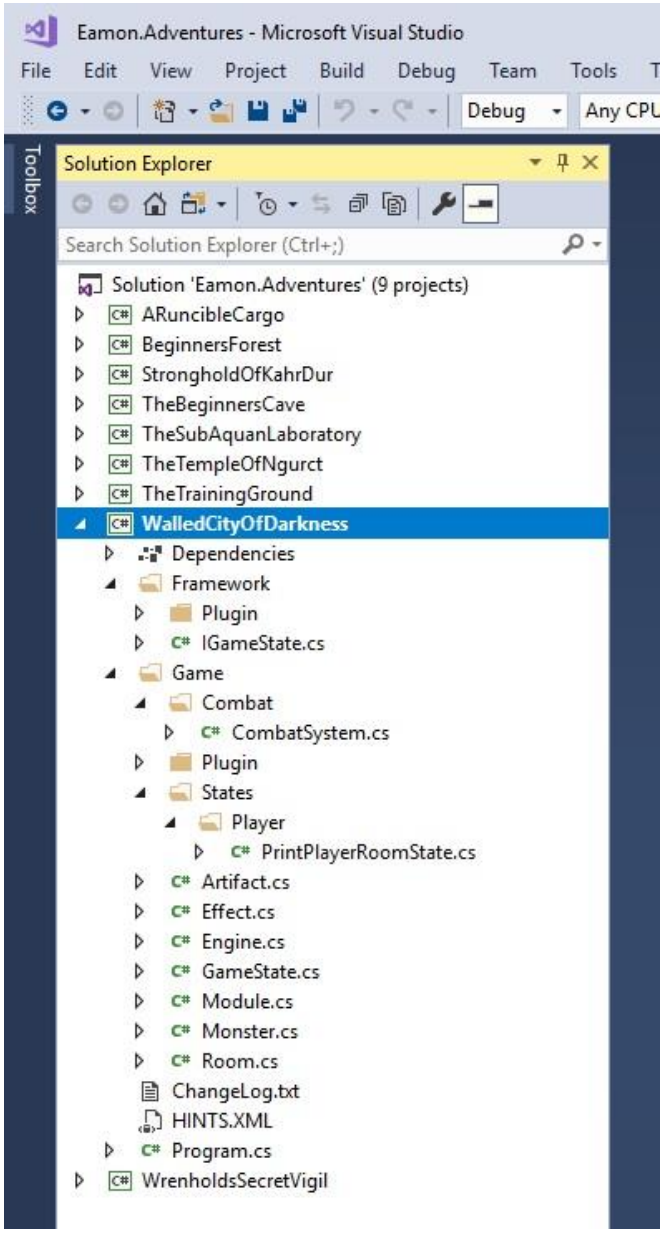
You can delete custom adventure classes just as quickly as you add them using the Adventure Support Menu. There are many reasons you may wish to do this. When you create a new adventure via the Add A Custom Adventure menu option, the system generates a complete set of foundational classes for it. But you will often find that you don't need all the classes produced. A generated class is only useful if you plan on overriding methods or properties in it; otherwise, it is just extraneous code bloat. There may be other times you are implementing some unique game behavior when you run into trouble and decide to restart the work or scrap it entirely.

Regardless of the reason, you can use the Delete Custom Adventure Classes menu option to delete one or more classes or interfaces from the game. If you have classes that are reliant on explicitly deleted interfaces your game will not compile until the classes are fixed or removed. If you delete classes, any paired interfaces will automatically delete as well. If the removed class has a corresponding .XML file (e.g., Room.cs and ROOMS.XML) it will be updated to reflect the appropriate parent class. There is no undoing this operation so you should back up your work if there is a chance you'll change your mind.

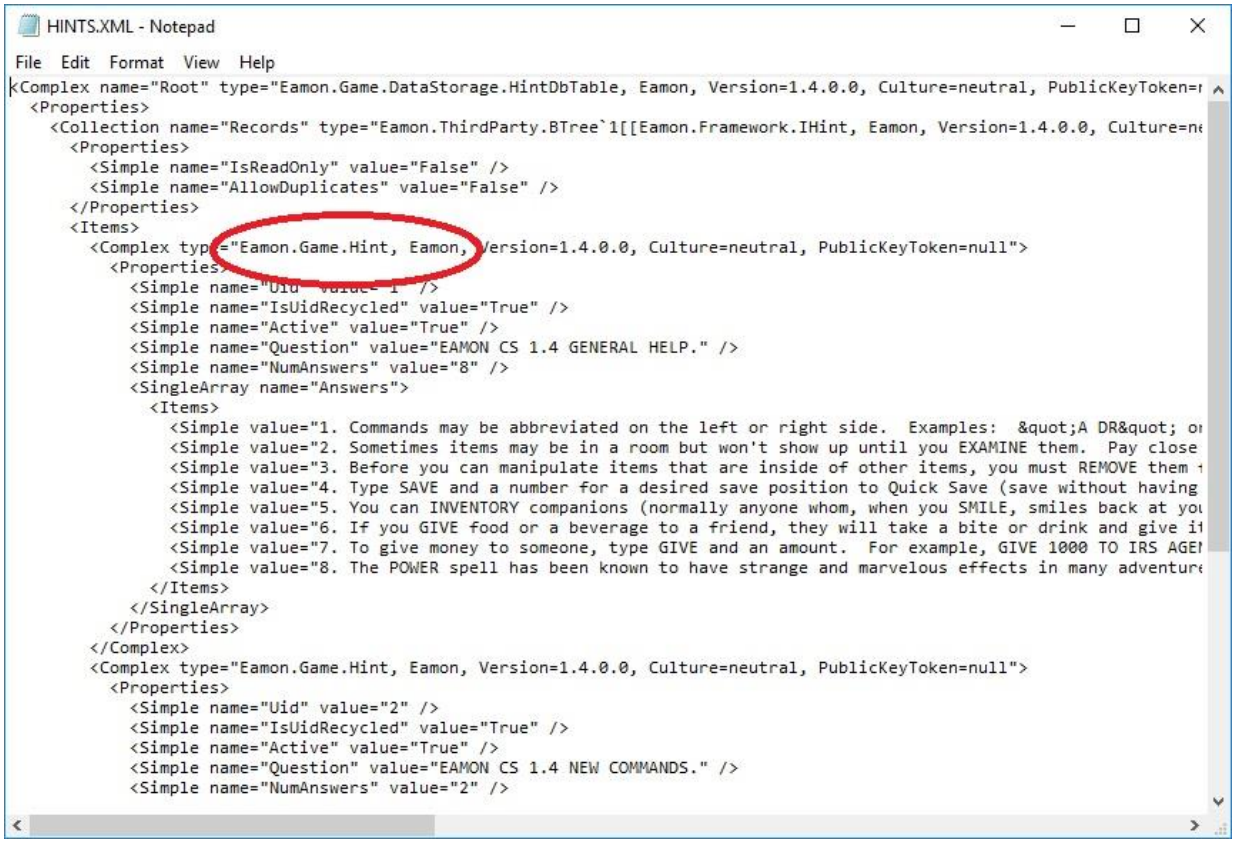
The following example demonstrates the removal of several classes from the Walled City of Darkness adventure.

1. You have several options when using this menu item. You can delete classes, interfaces, or both. File paths are specified relative to the adventure folder.

2. The classes selected for deletion are no longer in the project, and their matching interfaces are gone as well.



3. One of the deleted classes, Hint.cs, has a matching HINTS.XML file. Notice that the data type is no longer WalledCityOfDarkness.Game.Hint but was reverted to the parent class Eamon.Game.Hint.



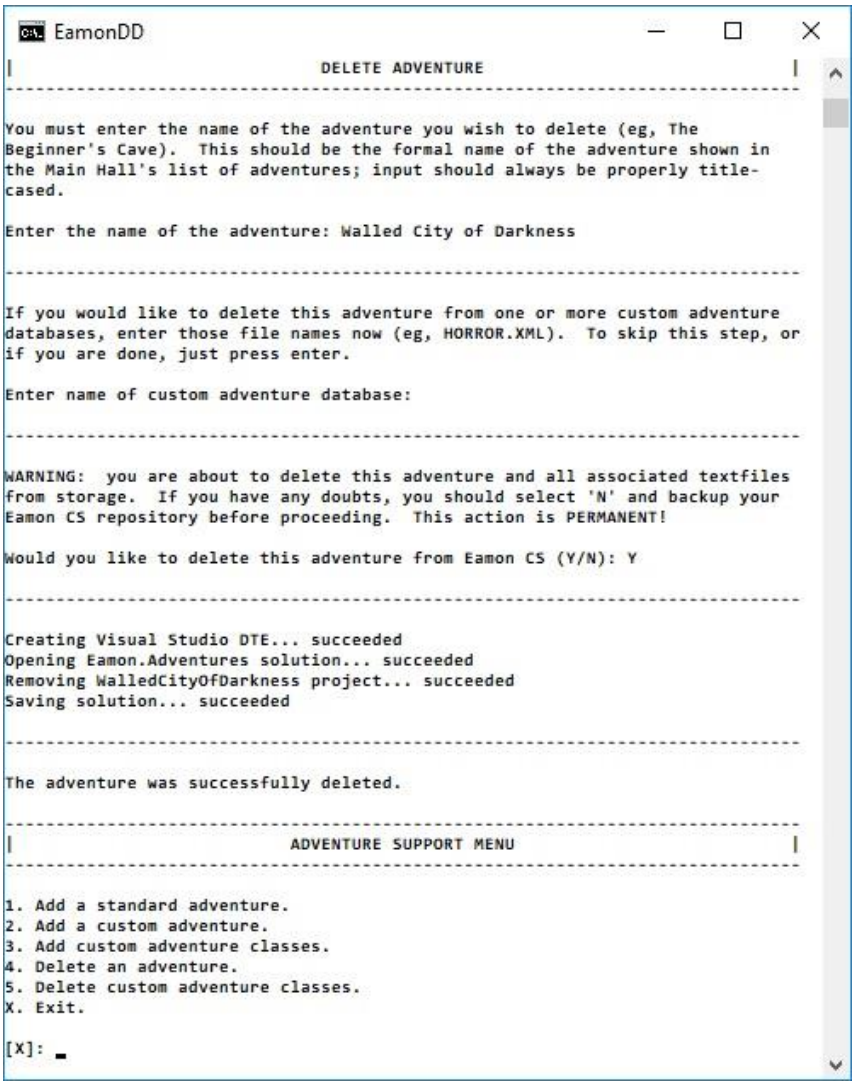
DELETING EXISTING ADVENTURES

You can also delete adventures from Eamon CS if the need arises. Maybe you were experimenting with a game scenario that didn't work out or have fully played through a title and want to make space on your file system. Perhaps you don't like an adventure and want to purge it from your Eamon CS repository. Whatever the reasons, the Adventure Support Menu has an option to do game deletion. Choosing this menu option will cause the system to prompt you for some key pieces of data and verify that you want to proceed. If so, it will completely remove all traces of the adventure from your Eamon CS system by doing the following:

- Deletes game library/binary files (Custom adventures only)
- Removes the game project from Eamon.Adventures.sln (Custom adventures only)
- Removes the game from appropriate Adventure database(s)
- Leaves the game folder under Adventures intact (but you can manually delete it yourself)
- Deletes the game QuickLaunch .bat and .sh files

If you delete a custom adventure in Unix, you will have to remove the game project manually from Eamon.Adventures.sln since Visual Studio integration is not available on that platform.

As you might expect, once a game has been deleted its data is not recoverable by normal means.


EamonDD
DELETE ADVENTURE
You must enter the name of the adventure you wish to delete (eg, The Beginner's Cave). This should be the formal name of the adventure shown in the Main Hall's list of adventures; input should always be properly title-cased.
Enter the name of the adventure: Walled City of Darkness
If you would like to delete this adventure from one or more custom adventure databases, enter those file names now (eg, HORROR.XML). To skip this step, or if you are done, just press enter.
Enter name of custom adventure database:
WARNING: you are about to delete this adventure and all associated textfiles from storage. If you have any doubts, you should select 'N' and backup your Eamon CS repository before proceeding. This action is PERMANENT!
Would you like to delete this adventure from Eamon CS (Y/N): Y
Creating Visual Studio DTE... succeeded
Opening Eamon.Adventures solution... succeeded
Removing WalledCityOfDarkness project... succeeded
Saving solution... succeeded
The adventure was successfully deleted.
ADVENTURE SUPPORT MENU
1. Add a standard adventure.
2. Add a custom adventure.
3. Add custom adventure classes.
4. Delete an adventure.
5. Delete custom adventure classes.
X. Exit.
[X]: