This note explains how to debug VaM plugins with breakpoints, step-by-step execution and variable inspection.

The C# interpreter itself could be modified in a way to to provide Soft Debugger capability. It means that upon running the interpreter would open an TCP port and let anything what is connected to the port to debug the program being executed by interpreter
(https://www.mono-project.com/docs/advanced/runtime/docs/soft-debugger/)

To connect and debug I was using dnSpy. Seems like you also can use MonoDevelop or a plugin for VS Code.

So, it short, you need two things:

1. dnSpy , which is a free program. You can download it from here (https://github.com/dnSpy/dnSpy/releases) or build it from sources.
2. Patched mono.dll

Seems like couple of yeas ago patched mono.dll was available for download on dnSpy github, but not anymore. You can still find links to the empty pages where mono.dll was . These days the only thing available is the instruction on how to compile mono.dll yourself.

https://github.com/dnSpy/dnSpy-Unity-mono
This tool provides a patcher which constructs source code for mono.dll. The accompanying README.md is quite cryptic and incomplete. More complete version could be found here:
https://www.unknowncheats.me/forum/unity/394581-patching-mono.html
, it is still hard to follow for my taste. My instruction follows:

So, lets start

1. I would assume the working folder is d:/prj and you have more than 10Gb of free space available.
2. I assume you have git installed and it is configured to work from any folder
3. From the directory d:/prj clone dnSpy's mono patcher:
   git clone https://github.com/dnSpy/dnSpy-Unity-mono.git
4. Also clone mono from Unity's repo:
   git clone --recurse https://github.com/Unity-Technologies/mono.git
5. make copy of umpatcher: copy folder d:\prj\dnSpy-Unity-mono\src\umpatcher to be d:\prj\umpatcher . Do not compile it in place. Do not modify cloned git repos in any way.
6. Open D:\prj\umpatcher\umpatcher.sln with VS2019 and Build Solution . Lets assume you have build Debug version of it and the exe appeared at d:\prj\umpatcher\umpatcher\bin\Debug\net48\umpatcher.exe
7. The original instruction discusses how to determine unity version and its mono version. You may want to do that for other games, but for VaM I would just list known values: Unity is 2018.1.9 and mono version is whatever b4d827005179f19e15114c02b564b448e38e8551 githash takes you to.
8. You need to set your git name and git email. Umpatcher seems to do some weird merge and it will fail if name and email are not set. To check if the name is already set, do
   git config --list
   and look for user.name and user.email lines. It those are not there, do
   git config --global user.name "My Name"
   git config --global user.email "myemail@example.com"

9. now run umpatcher:
   d:\prj\umpatcher\umpatcher\bin\Debug\net48\umpatcher.exe 2018.1.91 ae4c17e9c4fc0c648db0d19be1c028bab1a10dc1 "d:\prj\mono" "d:\prj\dnSpy-Unity-mono"
   do not use 2018.1.9 for a name. It will fail with an error.
   umpatcher runs for every version of Unity and checkouts different mono version into each folder. We don't need all that, but this is the way to trigger it to make desired sln file.
10. **In case umpatcher fails:**
    Umpatcher is not an idempotent tool: when run twice it will fail second time. If it fails halfway, you need to revert changes it made. So, before running umpatcher again, run:
    cd d:/prj/dnSpy-Unity-mono
    rmdir /s /q unity-2018.1.91
    git checkout master
    git reset --hard b4d827005179f19e15114c02b564b448e38e8551

    It is useful to make a .bat file with these lines if you wish to debug umpatcher.
11. **If umpather finishes succesfully** it creates sln files
12. Open d:\prj\dnSpy-Unity-mono\dnSpy-Unity-mono-v2018.x.sln with Visual Studio 2019
13. Choose Release_eglib for target and x64 for platfom
14. Build !
15. Go d:\prj\dnSpy-Unity-mono\builds\Release_eglib\unity-2018.1.9\win64 to pick up mono.dll

VaM

Now copy that mono.dll to VaM/Mono/EmbedRuntime . Do not forget to back up the original file.

Run VaM. Here you may want to check that mono indeed have opened tcp port 55555 at loopback:
netstat -a -b -n -p tcp

Run dnSpy

In windows file explorer open VaM\VaM_Data\Managed and locate Assembly-CSharp.dll . Drag and drop it into the left pane of dnSpy.

In dnSpy Debug → Start Debuging…

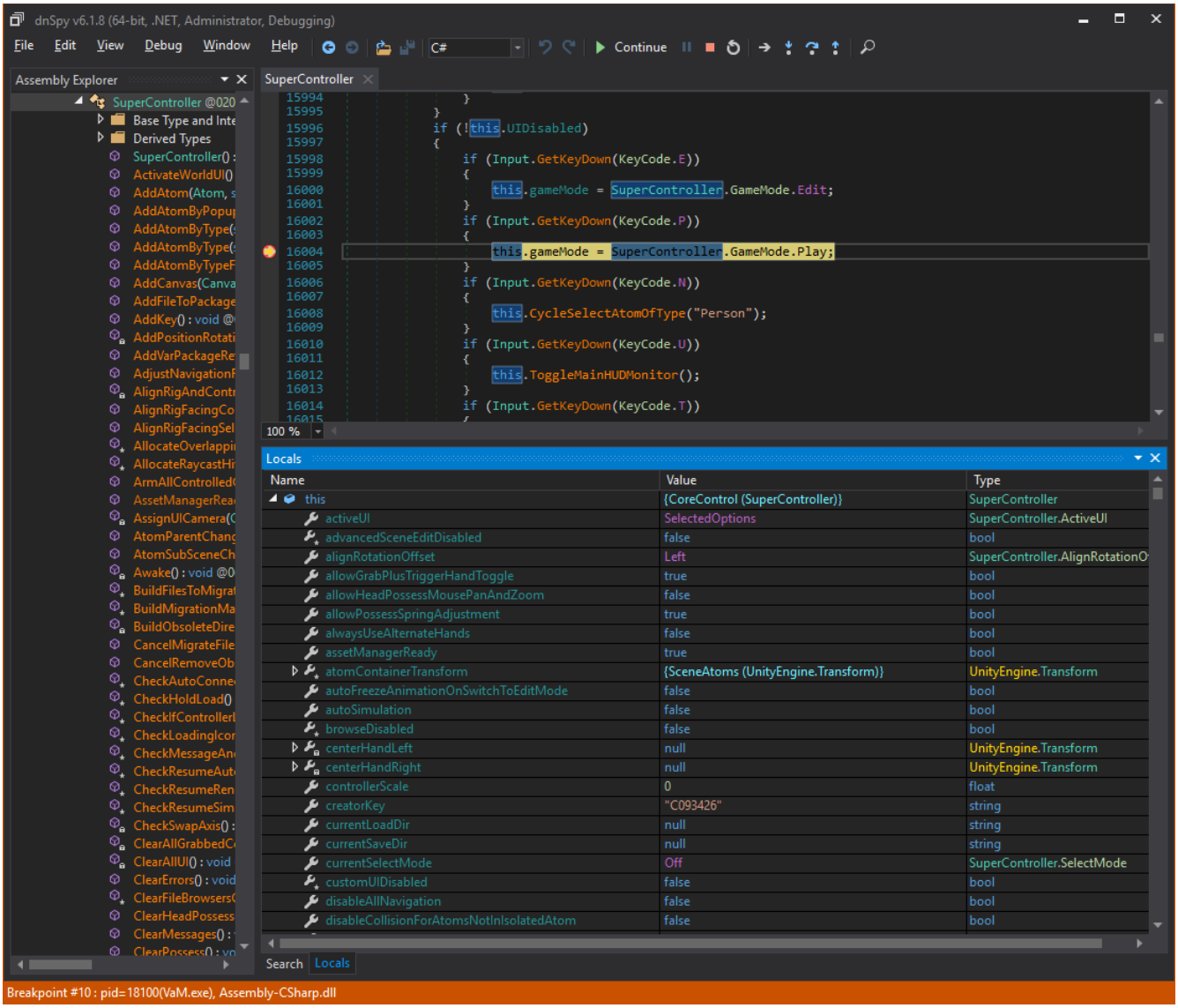
Select Unity (Connect) for a debug engine. Leave IP address empty and port 55555.

Click ok

Now lets make the first breakpoint. For beginnings lets make it in VaM itself. In the left pane unroll Assembly-CSharp and find default namespace denoted with single dash. It is at the very top of the list. Find SuperController and unroll. In the right pane press Ctrl+F to search and type "KeyCode.P". Look at line "this.gameMode = SuperController.GameMode.Play;" and set breakpoint by clicking on the gray strip at the left of line numbers.

Now switch to VaM windows and press key "P". VaM will freeze.

Now, this is how it should look like:



You can do step-by-step debugging and see local variables.

Plugins

Plugings are compiled at run time, so there are no place to put breakpoint to.

This would not do anything:

```
if(System.Diagnostics.Debugger.IsAttached)    System.Diagnostics.Debugger.Break();
```

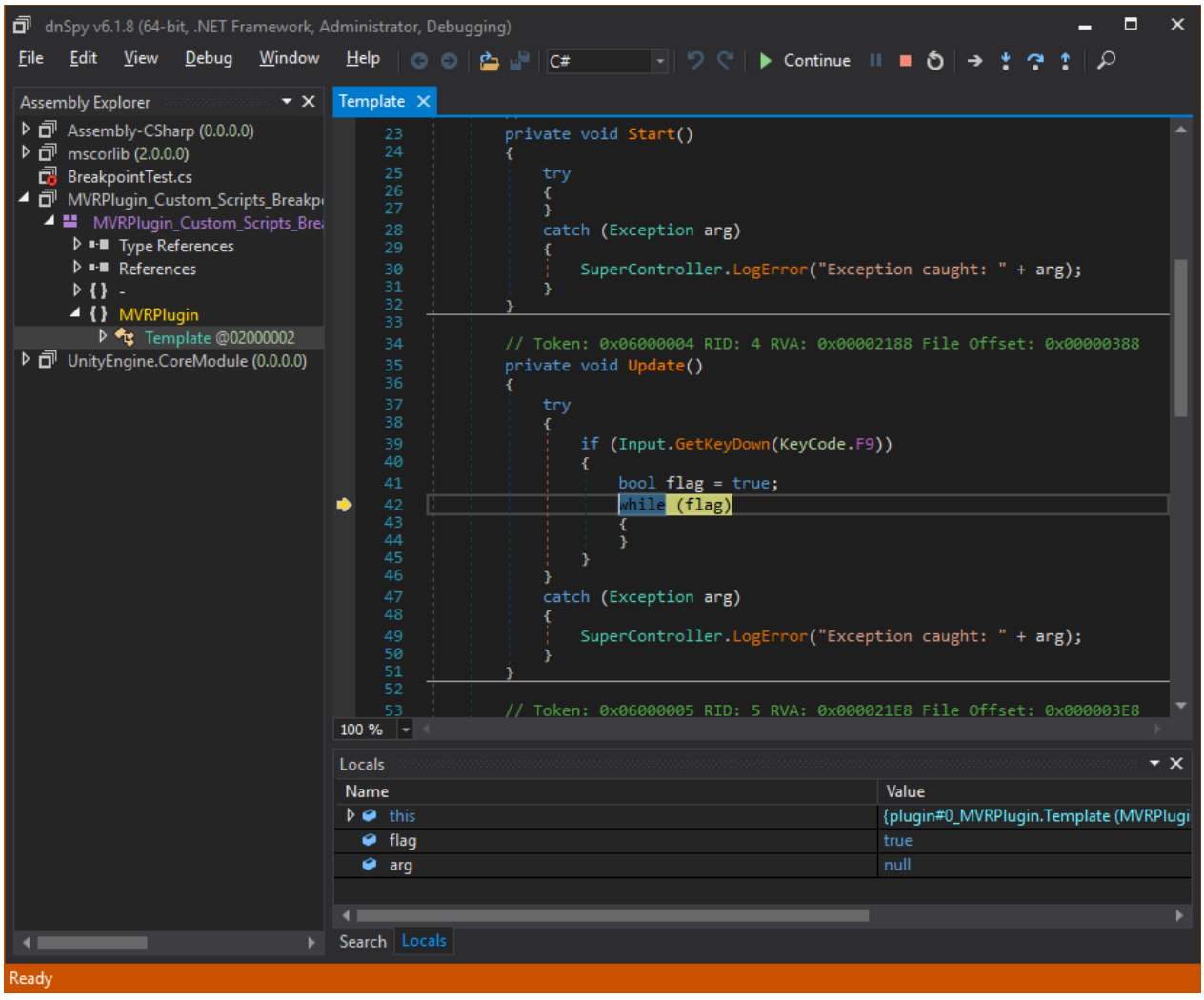
And just this would trigger VaM to terminate:

```
System.Diagnostics.Debugger.Break();
```

My idea is to trigger infinite loop by keypress and pause the program. Put this into the beginning of an Update function of a plugin:

```
if (Input.GetKeyDown(KeyCode.F9))
{
      bool stopFlag = true;
      while(stopFlag);
}
```

Drop the plugin .cs file to the left pane of dnSpy. Run that plugin and press F9. VaM will freeze. Switch to dnSpy and pause the execution: From here you can put up breakpoints. Change flag to false to continue running VaM.



dnSpy

Sometimes dnSpy will be unable to find the main unity thread. Sadly dnSpy does not have a tool to select a thread manually. The solution is to build dnSpy from sources with a little patch:

git clone --recurse https://github.com/dnSpy/dnSpy.git
then find dnSpy\Extensions\dnSpy.Debugger\dnSpy.Debugger.DotNet.Mono\Impl\
DbgEngineImpl.Threads.cs

,inside find IsMainThread and replace it with

```
bool IsMainThread(ThreadMirror thread) {
	if (alreadyKnowsMainThread) return false;
	string? methodName = null;
	try {
		var f = thread.GetFrames();
		methodName = f[0].Method.Name;
	}
	catch { }
	if (methodName == "Update") {
		alreadyKnowsMainThread = true;
		return true;
	}
	else return false;
}
```

then build