2-D Platformer Cloud Build Assignment

For more information on running the build tool please refer to the README at https://github.com/quinncheung/2d-platformer-build.

Language selection

The first design consideration for implementing this tool was whether to create a compiled binary application in a language such as Java or C# or other, or write a script in Python, Perl etc. The following criteria was considered to decide this:

- What is the target environment under which it will be run?
    - The most likely scenario is that the user trying this tool will be doing so on a Windows desktop, or at least can very easily access a Windows desktop OS.
    - Since there is no explicit requirement for the tool to be cross-platform, the selected programming language should be optimized to run under Windows.

- What is the level of complexity and logic for the software design and code?
    - This is a simple application that only requires a linear set of steps to automate what a user could do manually from a command line. There is not much logic required other than verifying the outcome of each step prior to executing the next step.
    - So many features offered by compiled languages such as object-oriented design, dynamic memory allocation and garbage collection are not required in this case. A procedural development approach is sufficient for this application.

- Which approach requires the least dependencies to be installed in the environment under which it should be run (i.e.: Windows, as described above)?
    - Obviously, a Java application requires Java to be installed, and C# also requires the correct version of the .NET library runtimes in its target environment.
    - Scripting languages such as Perl, Python, Ant, JavaScript also require libraries and an interpreter in order to run.
    - PowerShell also requires an interpreter, but PowerShell is shipped with most versions of Windows. Thus, under a Windows environment it is the most “out of the box” and stand-alone solution. The only restriction is the version of PowerShell being used.

Another criteria used for selection was: Which language would be the fastest for the developer to implement a solution? It was decided to use PowerShell.

Implementation

The software design is a procedural sequence of steps to build the Unity project deploy its artifacts. The steps are:

1. Remove any local copy of the 2-D Platformer project already on disk. This ensures that we start from a “blank page” and build the project from scratch.

2. Remove any local log files and other artifacts from previous builds.
3. Clone the 2-D Platformer project from the Git repository.
4. Call the Unity Editor with the appropriate command line arguments to build the project headlessly.
5. Zip the output of the Unity build into an artifact.
6. Zip the log files into an artifact.
7. Upload the artifacts to the JFrog Artifactory cloud storage.

Error Handing

At each step listed above, error handling is done to verify the success/failure of the step before running the next one. A try…catch block is used to halt the processing as soon as a step fails. The progression of the build is shown to the user via messages printed to console.

Most of the steps involve running third party applications: git.exe, Unity.exe, jfrog.exe. Thus, the script relies on the application's exit code to determine the success/failure of its step.

Other sources of failure include calls made to PowerShell modules with invalid arguments. For example,

- Trying to add files or folders that do not exist to an artifact
- One of the third-party tools listed above is not found.

PowerShell has good error handling and will throw an exception in these cases. The flow will end up in the script's catch block and the exception message generated by PowerShell clearly describes the problem.

Trade-offs/Compromises

Since it is written in PowerShell, this script is not cross-platform. There are different possibilities for addressing this:

- Make use of containers to perform the "build" step on other operating systems
- Re-write the script in a cross-platform language such as JavaScript
- Use a cross-platform automation tool such as Jenkins instead of a stand-alone tool.

Performance

The build status is displayed on the console with timestamps, and we can see that the compression of the Unity build output to a zip file is what takes the most time. This step is done using PowerShell's built-in compression function. It would be worth trying a third-party tool such as 7Zip to see if it is faster.

Overall, PowerShell is also likely not the fastest performing solution. For this particular application it is acceptable but If extremely low-latency would be required for the script interpretation then use of a different scripting language or a compiled language should be considered.

Improvements

Other sources of improvement could include:

- Showing a % complete during the Unity build – would indicate to the user that the system is not “hanging”.
- Automatically sending notifications to the stakeholders of the build result/status.
- Parameterize the build with a properties file or command line args for the build target, project to be built, version of Unity to use for build, artifact names and versions etc.