PROJECT MICRO

Technical Design Document

TINKERTAINMENT

Version 1.0

1 Document Changelog

Version	Description	Requestor	Date
0.1	**Taylor Bishop:** Creating the document	Stringer, Nausha, Ouellette	7/1/2016
0.2	**Taylor Bishop:** Filling out sections	Mario Rodriguez	7/3/2016
0.3	**Andrew Curley**: Comments and suggestions	Taylor Bishop	7/6/2016
0.4	**Taylor Bishop:** Added memory budget estimates, asset size / count limitations, and SteamVR risk	Mario Rodriguez	7/6/2016
0.5	**Taylor Bishop:** Wording	Andrew Curley	7/6/2016
0.6	**Taylor Bishop:** Incorporated Game Designer & Producer feedback	Clay Howell, Mario Rodriguez, Andrew Curley	7/6/2016
1.0	**Taylor Bishop:** Fixed headers, table of contents	Mario Rodriguez	7/6/2016
1.1	**Taylor Bishop:** Updated the Nightly Builds section to reflect recent changes and research	Mario Rodriquez	8/24/2016
1.2	**Jeremy Hicks:** Adding Minimum System Requirements section	Taylor Bishop	12/5/2016

2 Table of Contents

(Navigate to the bar on the left-hand side and click headers to jump to sections)

3 Project Programming Goals

3.1 Scope

The scope of the game is broken out into two separate tiers, the first being the minimum viable product shipped at the end of the development cycle, and the second being the ideal implementation that the team is striving for. The game is “complete” when the team meets the Tier 1 description.

Tier 0 - Minimum Viable Product

Players load up into a main menu level with objects representing “start game” and “exit game” in front of them. Players overlap their controllers with the objects and hit a button to activate them.

In the game, a table in front of players serves as the playspace. The background environment consists of simple geometry representing a kitchen environment. Players interact with objects to set a path for their mouse, activate the mouse, then find their way to the goal. The player’s goal is to get from point A to point B without having to switch out of the mouse mode - no enemies or patrols exist in the level for the player to interact with.

Tier 1 - Ideal Implementation

The player starts the game and is shown the main menu level. The main menu shows three tables, one in front of the player and tables on the left and right side of the player. There is a door behind the player which they can open to exit the game.

The table in front of the player in the main menu consists of a radio to adjust the volume, where they must activate the volume knob and physically turn the controller to adjust the volume.

The table to the right serves as the level select. A mouse sits in front of different plates. Each plate has a name tag describing which level it takes the player to and shows the number of stars the player achieved last time they played the level. Any levels not completed show empty plates.

The table to the left serves as the credits screen. Everyday objects represent each developer on the team. These objects have labels (similar to the level select labels) that show each developer’s name and specialization. The player can pick up and throw these objects around the main menu room.

Walls enclose the environment around the main menu to keep the player in a small area. There is a ceiling on the room with no windows, and the only way to leave is from the door behind the player.

Each object on the level select screen leads to a new level. The environment around each level consists of simple geometry representing a kitchen. The area where the level resides is colored.Players interact with objects to build pathways for the mouse. While in mouse mode, the player finds marbles and tacks that can be picked up and used to knock over or defeat enemy patrolling toy robots. If the mouse is caught by a patrol, the player must restart the level, but the objects placed in human mode stay where they were.

4 Principal Risks

This section of the document describes risks inherent to our development process and ways to mitigate them.

4.1 Platform Risks

4.1.1 Inexperience with VR game development

Risk - Our team has never developed for Virtual Reality (VR) before, and there are inherent risks that come with developing for new tech. The biggest risk this entails is lower velocity in task completion - tasks take longer to complete because the common VR pitfalls are unknown and the team has not experienced common implementations (sound, controls, motion, etc.) within a VR environment.

Impact - This risk has a medium impact, as issues with our newness to the technology may slow development.

Mitigation - Every aspect of the game must be touched in pre-production to ensure that implementation is feasible and works as expected before full production begins. The team needs to toy with lighting to make sure it functions similarly in VR, 3D audio needs to be tested in some way, and all mechanic functionality needs to be in as soon as possible to find the fun and prove that we can build out on the foundation of the game.

4.1.2 Limited Number of Vive Headsets

Risk - Limited hardware poses a pipeline risk, because team members need to be able to test their assets in-game in the headset, and cannot do so while it is in use by another teammate.

Impact - This risk has a medium impact, as the limited number of headsets may slow down development and team members may be waiting to test their assets.

Mitigation - The team must build testing into their department pipelines. Dedicated machines for each headset are set up by the end of pre-production. A Keyboard Actor pawn is used to test mechanics functionality, so that dummy controllers can be used with a mouse and keyboard on the development laptops - reducing the amount of micro-testing among the LD and SD departments.

4.1.3 The dedicated machines hosting the Vives go down or have problems

Risk - The dedicated machines that the Vives use might have some error(s) and cannot be used for a period of time during development.

Impact - This risk has a high impact, as the team cannot test the game and assets until the dedicated machines come back online.

Mitigation - The Lead Artist, Lead Programmer, and Lead Level Designer use their development laptops as the VIVE dedicated machines for their respective department. This restricts them from doing work while the department uses their computers for development, but alleviates the inability to test assets and the game in the headsets.

4.1.4 SteamVR Automatic Updates

Risk - SteamVR automatically updates when you launch it, and may break the ability to test the game in-editor.

Impact - This risk has a high impact, as assets developed cannot be playtested or checked for accuracy in the headset. Results in lost time and uncertainty in the team on whether their assets work or not.

Mitigation - A specific version of SteamVR, 1464744840, must be loaded on the dedicated Vive machines at all times. The team will freeze the SteamVR version until development ends. Loading an older version of SteamVR prevents it from updating, and fixes the issue of it automatically updating whenever a team member launches Steam or SteamVR.

4.2 Engine Risks

4.2.1 Maturity of VR support in Unreal 4

Risk - UE4 has only recently begun supporting VR and the Vive, and because of that, bugs remain in the engine. Hotfixes continually come out every few weeks for the engine, fixing these bugs. As an example, 4.12 Hotfix 3 fixed a crash that occurred when testing in the project with shader complexity view turned on.

Impact - This risk has a medium impact, as we may encounter engine bugs during development.

Mitigation - Engine hotfixes are continually installed on the Lead Programmers' home computer when they come out and the project tested on them. If the project runs as it did in the previous version and the hotfix is deemed to fix issues that we have / may potentially have, the project is upgraded to the latest version and all team members are instructed to update their engine installs to that hotfix outside of core hours.

4.3 Middleware Risks

4.3.1 Substance Plugin never used by team before

Risk - The team, outside of the artists, has never used the Substance Plugin before for UE4. The artists need it and all team members must have it in order for the project to work.

Impact - This risk has a low impact, as it may cause confusion or stop a team member from completing their work when they expected to.

Mitigation - If a problem occurs importing textures into the project among team members, team members must consult the Lead Artist on how to fix the issue, and how to use the Substance Plugin.

5 Performance & Resource Budget

On the Vive, everything must run at 90FPS at all times. That affords 11ms every frame to rendering and gameplay simultaneously.

The two places that are the most problematic for performance are **rendering** and **physics**. Most gameplay is centered on manipulating physics objects, and collision detection and physics updating is not cheap. Ensuring that we spend our time in these two areas and that we cut out extraneous calculations, gives us the most bang for our buck.

Below are texture-size and vertex count maximums for different assets in the game, as well as memory budget estimates. Keep in mind that memory constraints won't kill performance, **but do dramatically affect loading times.**

5.1 Do's and Do-Nots For Performance

- **DO NOT** use ragdolls, PHATS, or deformable physics meshes in the game, at all.
- **DO NOT** exceed the maximum lightmap count in a single level.
- **DO** use flat textures for objects and environment pieces, whenever possible.
- **DO** use static lighting instead of dynamic whenever you can.
- **DO** use extreme caution when doing anything with particle systems (Run it by the Lead Programmer if you are unsure if it is too much).

5.2 Memory Utilization Estimate

Asset	Memory Amount	Total Unique In-Game at One Time	Total Memory Amount
Mouse Character	1MB	1	1MB
VR Controller Meshes	1.5MB	1	1.5MB
Pickup-able Meshes	0.4MB	100	40MB
Static Meshes	0.1MB	100	10MB
Lightmaps	12MB	8	96MB
Audio Effects	2MB	30	60MB

5.3 Assets Budget Estimate

5.3.1 Meshes

Mesh	Average Tri Count	Average Vert Count	Max Tri Count	Max Vert Count
Mouse Character	10,000	4,000	12,000	5,000
VR Controller Meshes	10,000	15,000	12,000	17,000
Pickup-able Meshes	1,500	2,000	2,000	2,500
Static Meshes	400	500	600	800

//UPDATE THIS

5.3.2 Textures

Texture	Max Texture Size
Mouse Character Texture	2048x2048
VR Controller Texture	1024x1024
Pickup-able Texture	512x512
Static / Environment Object Texture	512x512

6 Naming Conventions

The team must prepend all files with the type of asset they belong to. The name follows the asset type, and after that is any modifiers it needs and the version number. Once a team member finalizes an asset and no changes are made to it, it is given the version FINAL. Version number is in natural numbers from 1 to number of iterations. Some examples follow:

MAT_Toaster, MAT_Toaster_RED_FINAL, MAT_Toaster_LOWRES_3, MAT_Toaster_LOWRES_FINAL

Prefix	Description	Example
MAT_	Materials.	MAT_Toaster_RED
TEX_	Textures.	TEX_Table_FINAL
ANIM_	UE4 Animations	ANIM_Mouse_3
BP_	Blueprints	BP_MicroPawn
BPLIB_	Blueprint Function Libraries	BPLIB_TeleportFunctions
BPI_	Blueprint Interfaces	BPI_PickupableInterface
WidgetBP_	Blueprint Widgets	WidgetBP_MainMenuHUD
CUE_	Audio Cues	CUE_ToasterFellOnFloor
PARTICLE_	Particle Systems / Effects	PARTICLE_ToasterTurnedOn
SM_	Static Meshes	SM_Toaster_RED_FINAL
SK_	Skeletal Meshes	SK_Mouse_ROUGH

7 Blueprint Coding Standards

A few simple rules govern the cleanliness, portability, and separation of our blueprints. Team members must follow these rules at all times.

RULE 1 - COMMENT BOXES

If functionality is not self-contained to a singular function or it resides in an Event Graph, then a comment box encapsulates that functionality, **no matter how few nodes the functionality contains.**

RULE 2 - FUNCTION LIBRARIES

All functionality in the character blueprint is broken into a function library, with a reference to the character passed in. This ensures that functionality can be built out in parallel without multiple programmers having to “pass the pumpkin” to each other for their implementations.

RULE 3 - NO SPIDERWEBS

Node links overlap **as little as humanly possible.**

RULE 4 - GO ACROSS, NOT DOWN

Functionality goes from left to right, then top to bottom, just like the English language. This means that node connections succeed each other to the right. Additionally, team members must not link **nodes between each line.**

IF YOU ARE CAUGHT VIOLATING ANY OF THESE STANDARDS, CLAY THE GAME DESIGNER GETS TO RUB YOUR HEAD FOR 10 WHOLE SECONDS, UNINTERRUPTED!

8 Interface Standards

These standards are a little looser and not as cut-and-dry as the blueprint coding standards, but they are just as important. The purpose of these standards is to ensure that as little work as possible goes into using the mechanics and blueprints the programmers create. It is to ensure that the level designers can use our mechanics as painlessly as possible.

THE PICKUPABLE INTERFACE

The pick-up mechanic implemented must rely on the pickup-able interface that we apply to objects players can pickup, place down, and throw across the room. This means that when an artist creates an asset, such as a toaster, all the level designers need to do is attach the pickup-able interface to it in order for players to be able to pick it up in-game.

THE INTERACTABLE INTERFACE

When players interact with an object, whether it be a radio to turn the game volume up and down, or a door where the knob can be grabbed on the door opened, an interactable interface must be applied to that object and the InteractWith() functionality overridden in the object to get the requested functionality. This means that if a player interacts with the door, the door blueprint that contains the model handles the opening, closing, and what happens when the door is interacted with instead of the character blueprint or something else.

9 Version Control

9.1 Perforce File Structure

This is the project folder structure in Perforce and in the UE4 project. Descriptions in red italics are for folders with non-obvious uses.

/Config *Contains UE4 configuration files - Source Control Settings, window layout, etc.*
  ...
/Content
  /Animation
    /Character
  /Art
    /Zoos
      /Dev
      /Mace
      /Nina
      /TaylorG
      /TaylorM
  /Audio
    /Music
    /SFX
  /Blueprints
    /CharacterBPs
    /FunctionLibraries
    /GameCommon
    /GameModeRelated
    /Widgets
  /LD
    /Zoos
      /Alex
      /Jacob
      /James
      /Michael
      /Sam
      /Steve
  /Levels
    /Maps
    /Streams
  /Materials
    /Character
    /Environment
      /Hero
      /Modular
      /Prop
  /Meshes
    /SkeletalMeshes
    /StaticMeshes
  /Particles
  /Programming
    /Zoos

/Ben
/Clay
/Jeremy
/Komal
/Taylor
/Textures
/Character
/Environment
/Hero
/Modular
/Prop
/Intermediate *Used by UE4 when packaging the project*
...
/Originals *Not inside UE4 project*
/Concepts
/Character
/Environment
/Meshes
/SkeletalMeshes
/StaticMeshes
/Animation
/Character
/Textures
/Character
/Environment
/Hero
/Modular
/Prop
/References
/Animation
/Character
/Environment
/ProjectManagement
/Saved *Used by UE4 to hold save data for the game and for the engine (Last files open, etc.)*
…

IF YOU ARE CAUGHT VIOLATING THE PROJECT FOLDER STRUCTURE, CLAY, THE GAME DESIGNER RESERVES THE RIGHT TO RUB YOUR HEAD FOR 10 WHOLE SECONDS, UNINTERRUPTED!

9.2 Perforce Setup Tutorial

The Perforce Setup Tutorial Link (Located on the Wiki)

9.3 Perforce Use Tutorial

The Perforce Tutorial (Located on the Wiki)

10 Development Tools

10.1 Automated Nightly Builds

- Every night, set at 12AM CST, a batch file (Located under **//depot/C25/TGP/Capstone/ProjectMicro/ProjectManagement/ProgrammingReferenceMaterial/NightlyBuild.bat** for reference) is run on one of the programming dedicated Vive machine (For the moment, the dedicated laptop in Rm 134) that will be available the next day.
- The build will be available every morning for the Publishing Producer, Producer, or Game Designer to playtest. He will then go to the Lead Programmer if issues arise with the build or framerate, and to the Game Designer if there are general comfort concerns.

10.2 Installable Executables

- Research will begin in POCG on creating an installer (An external source must be used to create the installer and link it to the project)
    - Implementation will begin in Vertical Slice, with a working installer by the end of the milestone
    - By Alpha, packages will be run through the installer on a regular basis to ensure that they are still functioning properly
- The Publishing Producer will copy the executable to a USB drive
    - They will uninstall the previous version obtained and install the new one, ensuring that it runs properly, contacting the Lead Programmer if issues arise

11 Technology Sources

11.1 Production

Software	Description
Airtable	In-browser spreadsheet manager that the team uses for the Asset Database. (https://airtable.com/shr4dDl0r7Y5c18Uo)
Google Docs	In-browser word processing software that allows for the creation of spreadsheets, word documents, presentations, and more. Used by the team for almost all documentation.
JIRA	Project management tool for the team's product and sprint backlogs. Additionally, this tool includes the bug tracking database.
Confluence	In-browser wiki software used by the Guildhall and the team to store easy-to-access links to all project documentation.
Steam	Valve's gaming platform that hosts SteamVR.
SteamVR	Used to turn the Vive headsets on and run the game properly through the headset.

Slack	Messaging software that the team uses to communicate and share links and incidental files with.
Unreal Engine 4.12.X	The Unreal 4 engine the team will be using for the duration of the project. The last number may change depending on which hotfix versions we upgrade to.
Microsoft Visio 2013	Tool used to create the departmental pipeline flowcharts.
Microsoft PowerPoint 2013	Presentation tool used for public presentations to the professors and stakeholders.

11.2 Level Design

Software	Description
Photoshop CC	Image-editing software used to create graphs, charts, and sketches.
Microsoft Word 2013	Used to create Level Design documents and other documentation.
Adobe Illustrator CC	Used to create graphs, charts, and sketches.

11.3 Art Creation

Software	Description
Photoshop CC	Image-editing software used to edit textures and create concept art.
3DS Max 2016	Modeling software used to create meshes, animations, and skin meshes.
Mudbox 2016	Used to create high-poly sculpts of models.
Substance Painter 2	Used to create model textures.
Substance Designer 5	Used to create materials used by Unreal 4.
Crazy Bump 1.22	Used to generate normal maps for model textures.

11.4 Programming

Software	Description
Visual Studio 2015	IDE for writing code and building the project. Is the default IDE that UE4 uses.
Sublime Text 3	Text editor with the capability for plugins. Friendly to programmers

Notepad++	Text editor with the capability for plugins and alternate viewing modes. Friendly to programmers.

12 Stress Test Results

12.1 Content and Corresponding FPS

Level	Maximum Tested Contents	FPS
POCT Env	100 Meowbots	90
POCT Env	100 Mouse Characters (with animation)	90
POCT Env	100 Hats	90
POCT Env	40 Static Lights	90
POCT Env	10 Dynamic Lights	75 - 90

NOTE: Frame rate drops after spending a long time in the Vive regardless of number of objects or lights.

12.2 Level-Associated FPS

Level	FPS
Main Menu	90 (occasionally drops to 75 FPS)
Lee01	90 (single drop to 75 FPS)
Lee02	90 (no FPS drops)
Lee03	90 (occasionally drops to 75)
Domino	75 - 90 (consistently drops to 75)

12 Maximum Content Allowed

12.1 Maximum Content

Content	High-Water Mark
Environment Meshes	30,000 tris
Pickupables	120
Low Poly-Count Static Meshes	140
High-Poly Count Static Meshes	60

12.2 Notes On Increasing Performance

1. **No dynamic lighting.** Use only lights marked “static” (No stationary either).
2. **Meshes** should be given the smallest, most primitive collision volumes possible.
3. **Meshes** should not have shadows baked in.
4. **Moving Objects** should not cast shadows.
5. **Try to limit** transparency as much as possible.

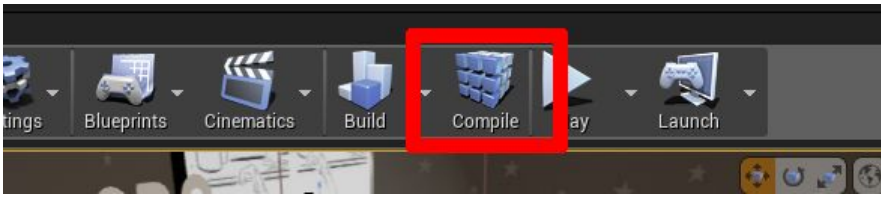
13 How to Fix Compilation Errors

13.1 Reasoning

This section of the document is aimed at Level Designers and Artists during split core hours when the programmers are not here. As programmers, our job is to make sure that **NO** compilation errors exist in the codebase Monday when split core starts. So if a compilation error occurs, it's because a couple things might need to be done on your side to fix them (Not a code / codebase problem)

13.2 Compiling

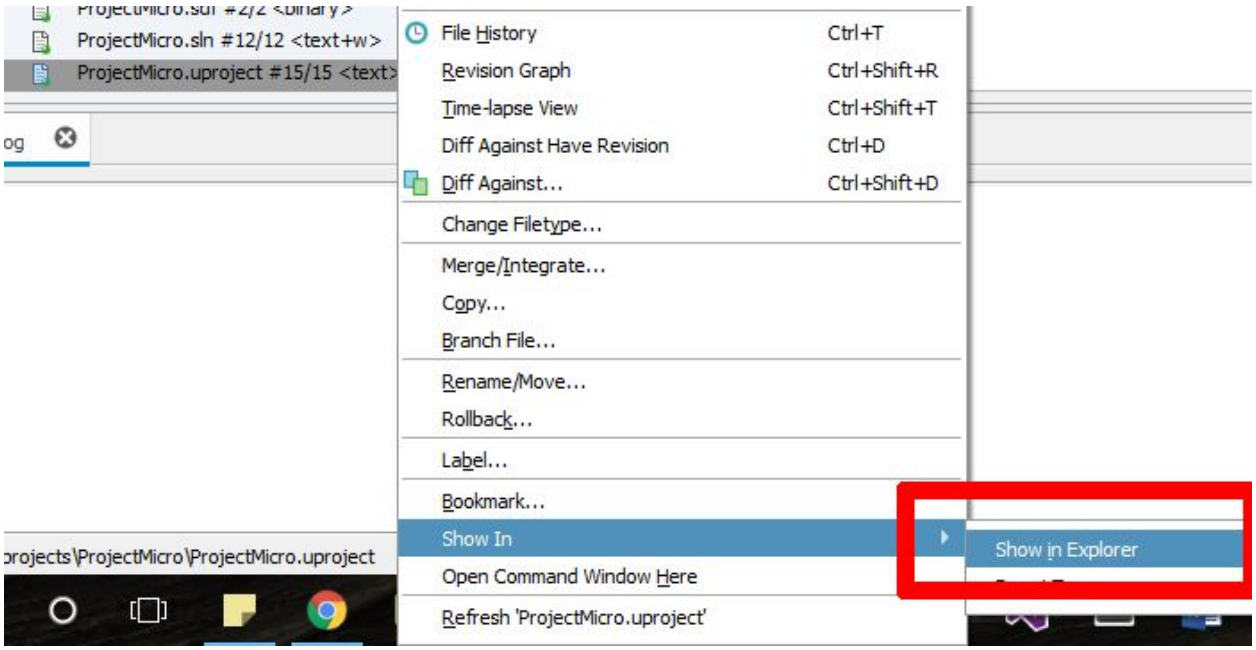
First of all, most errors can be solved by simply hitting the big Compile button in the toolbar:



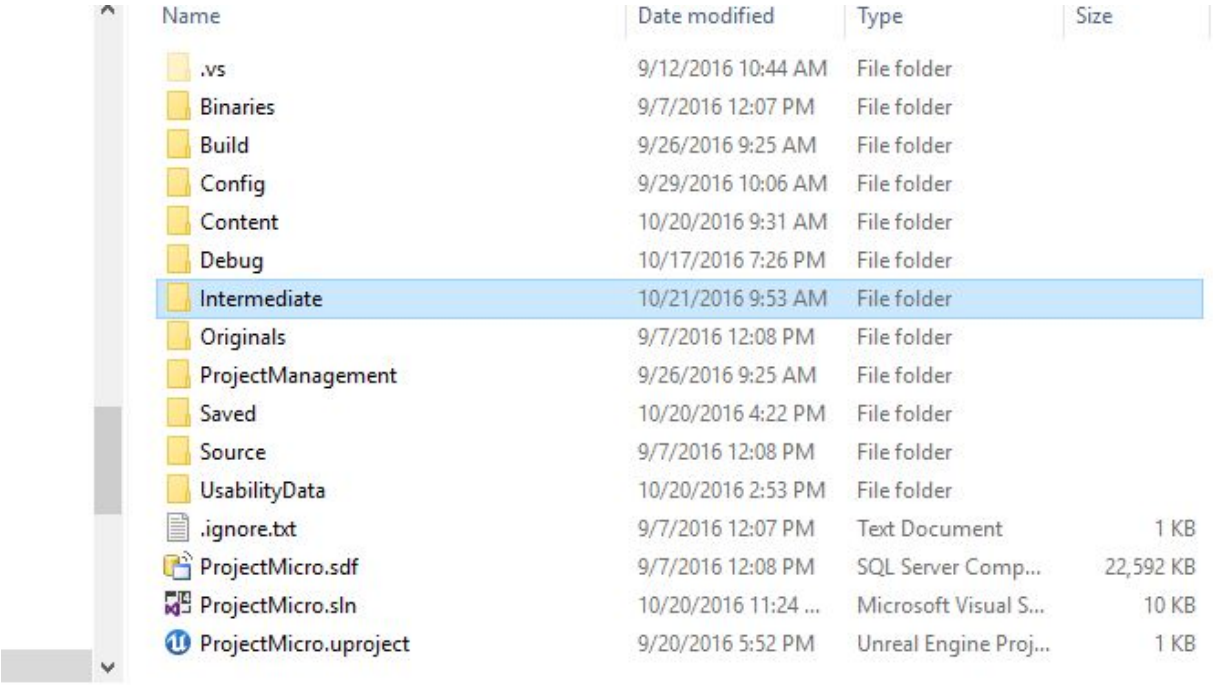
If the error continues to occur, you need to **regenerate the Visual Studio Project Files.**

13.3 Regenerating Visual Studio Project Files

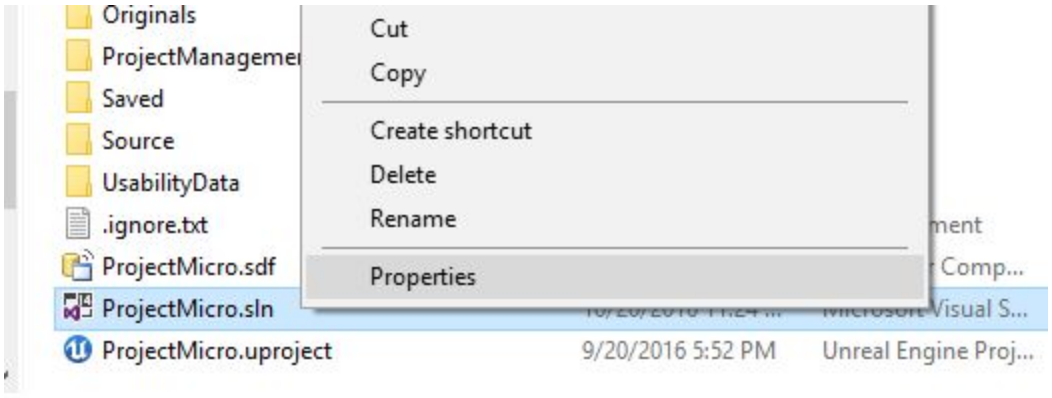
1) Close down the editor.
2) Open Perforce.
3) Right click the .uproject and click **Show In -> Explorer.**



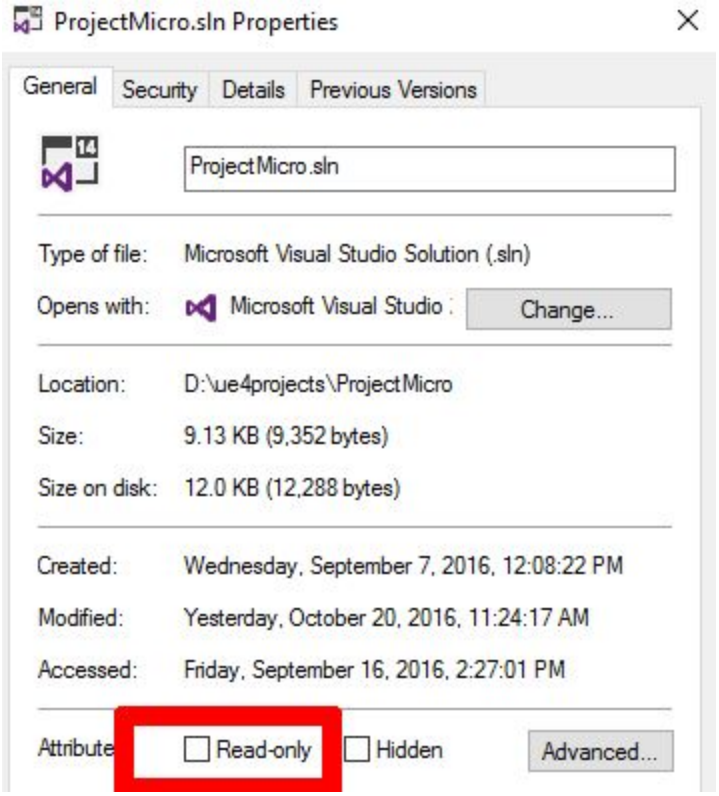
4) This will take you to the project folder. Delete the **Intermediate** folder.



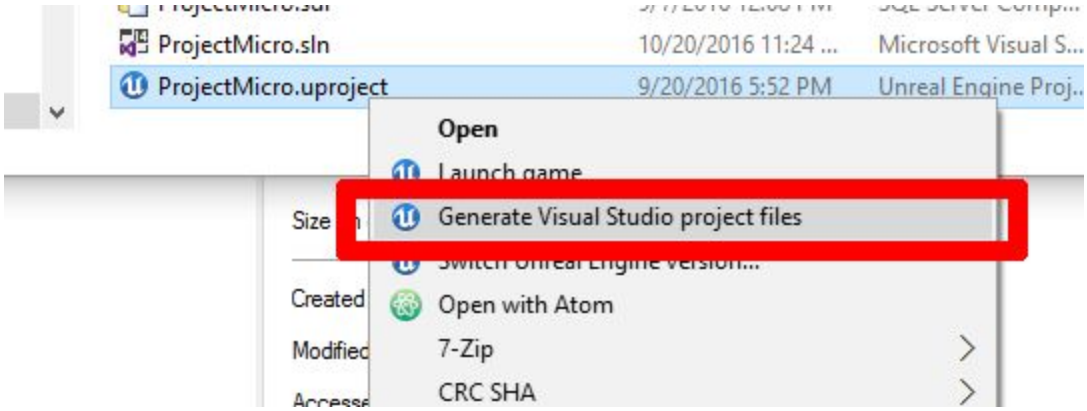
5) Right-click the ProjectMicro.sln file (Always the second file up from the bottom, you may not see the .sln extension name if you don't have extensions visible on file names) and click **Properties**.



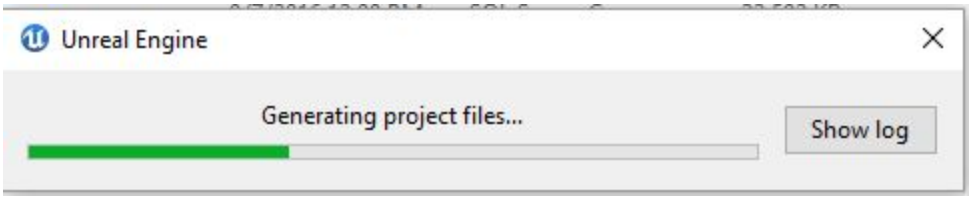
6) Make sure that **Read-Only** is **UNCHECKED.**



7) Hit OK. Now right-clock the ProjectMicro.uproject (Last file down in the project folder) and hit **Generate Visual Studio Project Files**



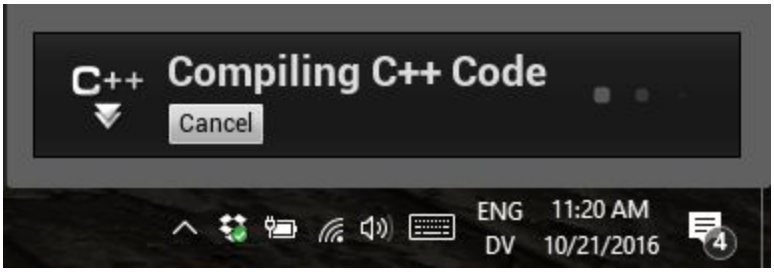
8) If you did this part correctly, then this window will pop up:



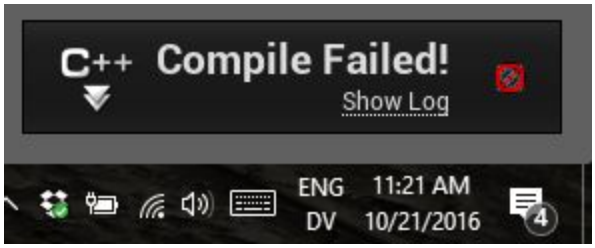
9) Now open up the project, hit the **Compile** button once more, and you should be good to go!

13.4 "Compile Failed!" After Regenerating Visual Studio Project Files

While compiling, you'll see a window like this in the bottom right-hand corner of the editor:



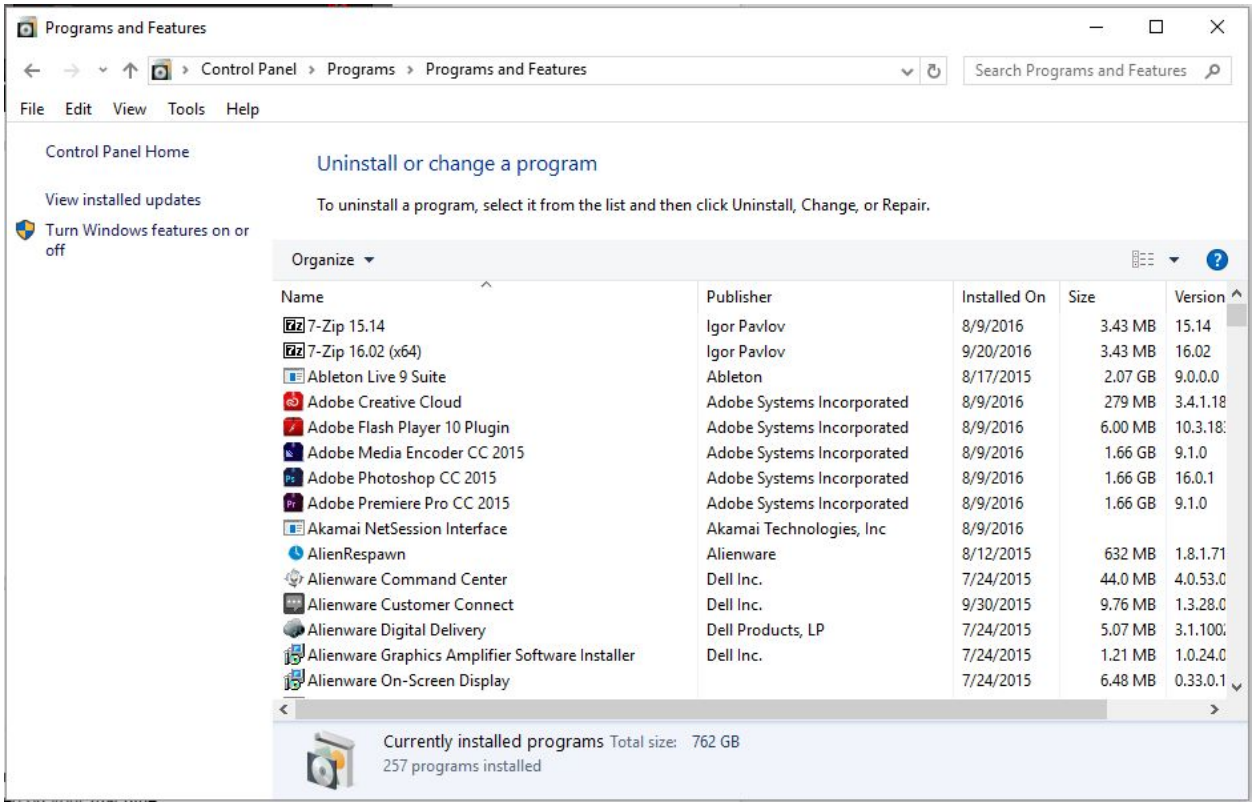
If the compile fails, you'll see a window like this:



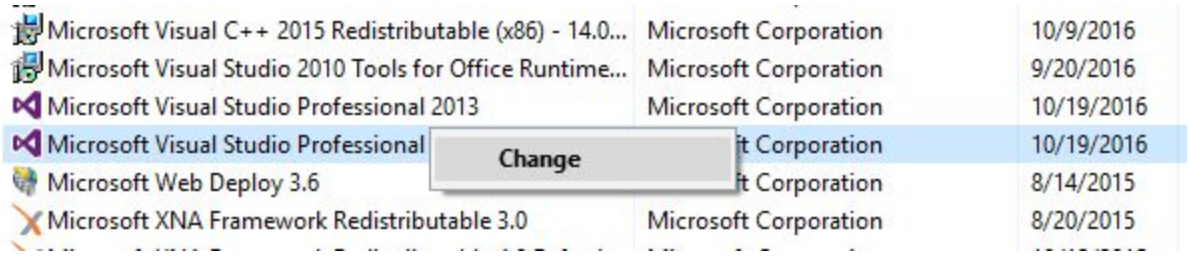
If this happens, you most likely need to **repair Visual Studio 2015**.

13.5 Repairing Visual Studio 2015

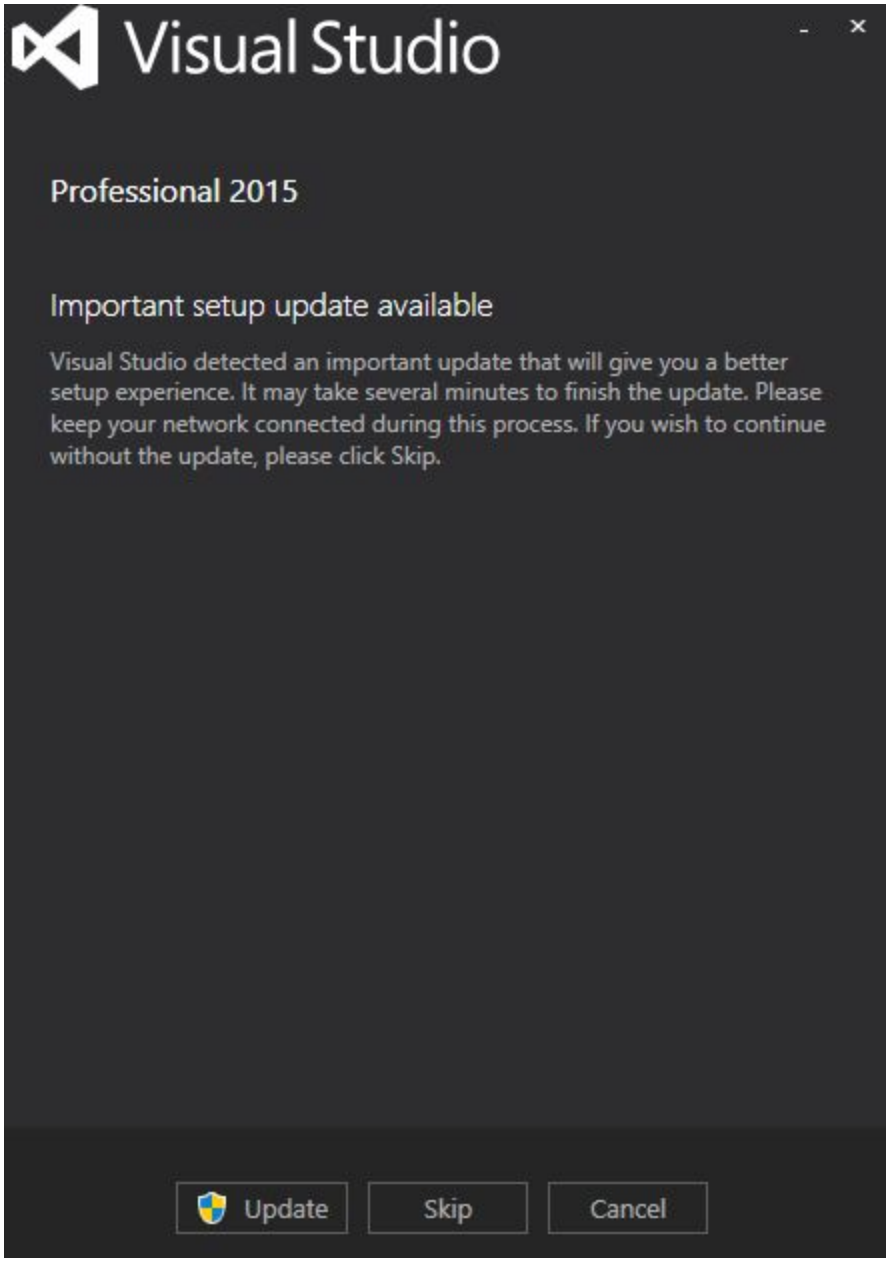
1) First of all, make sure you have either **Visual Studio Community 2015** or **Visual Studio 2015 Professional** installed on your machine.
2) Open the Control Panel.
3) Go to **Programs -> Programs and Features.** The window looks like this:



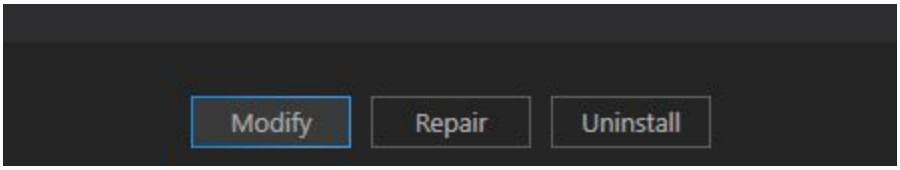
4) Scroll down to **Microsoft Visual Studio (Community / Professional) 2015**, right-click and hit **Change.**



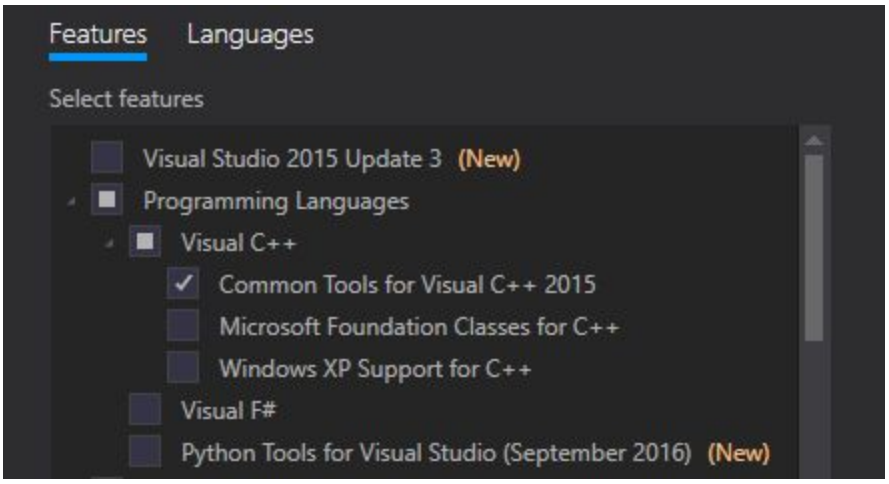
11) If this window pops up, hit **SKIP:**



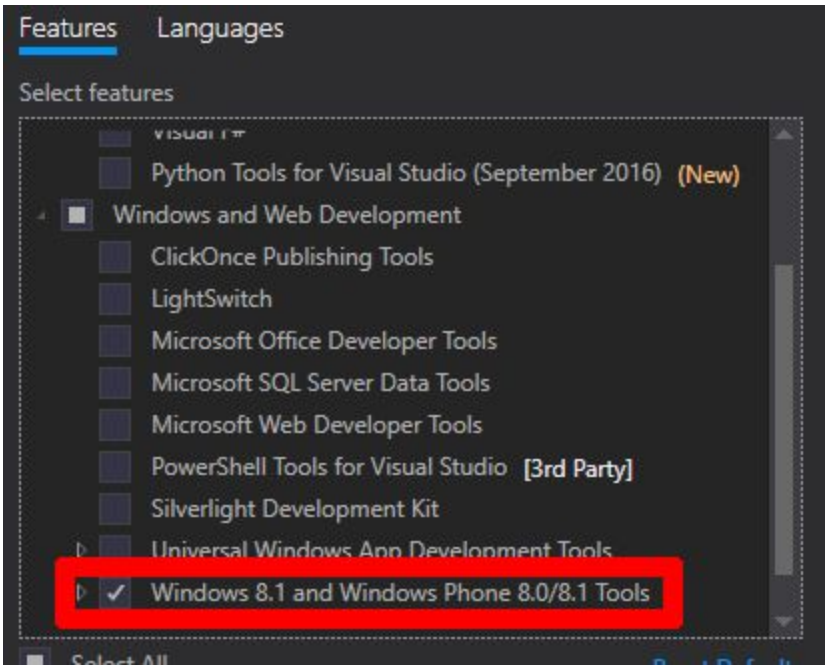
12) Make sure that Visual Studio is closed.
13) On the next window, hit **MODIFY:**



14) This will open up a **Features** window where you can modify plugins to Visual Studio. First, let's make sure that Visual C++ is installed:



15) Next, scroll down to **Windows and Mobile Development** and expand it.
16) Scroll down to **Windows 8.1 and Windows PHone 8.0/8.1 Tools** and make sure it is checked.



17) Hit **Next,** make sure **Selected Features** has something in it (That we're actually installing what we need), then hit **Update.**
18) As a warning, this update will take a few minutes to finish. Once it's done, exit out of that window, go to section **13.3** of this document to regenerate the Visual Studio Project Files, open the editor, hit **Compile**, and you'll be good to go!

14 System Requirements

Recommended

- GPU: NVIDIA GeForce® GTX 1080
- CPU: Intel® i7-6700
- RAM: 8 GB
- USB Port: 1x USB 3.0
- Operating System: Windows 10

Minimum Verified Playable

- GPU: NVIDIA GeForce® GTX 970
- CPU: Intel® i7-4770
- RAM: 8 GB
- Video Output: HDMI 2.0 (linked to Vive output), DisplayPort 1.2 (linked to monitor output)
- USB Port: 1x USB 3.0
- Operating System: Windows 10 (64-bit)

Noted the machine used with this spec was not new, but 3-4 years old, purchased in 2012-2013.

Vive Minimum Recommendation

- GPU: NVIDIA GeForce® GTX 970, AMD Radeon™ R9 290 equivalent or better
- CPU: Intel® i5-4590, AMD FX 8350 equivalent or better
- RAM: 4 GB or more
- Video Output: HDMI 1.4, DisplayPort 1.2 or newer
- USB Port: 1x USB 2.0 or better port
- Operating System: Windows 7 SP1, Windows 8.1 or Windows 10