

---

# GDBS/SVBS Midterm Project Overview

---

# Goal

---

- These courses are designed to familiarize students with the development process by implementing and completing a multiple month game project as a team.



# Who are we

---

Jason Hinders

Program Director:

[jhinders@fullsail.com](mailto:jhinders@fullsail.com)

Rebecca Leis

Department Chair:

[rleis@fullsail.com](mailto:rleis@fullsail.com)

Steve VanZandt

CD of AHI:

[svanzandt@fullsail.com](mailto:svanzandt@fullsail.com)

Rod Moye

CD of PP2:

[rmoye@fullsail.com](mailto:rmoye@fullsail.com)

John O'Leske

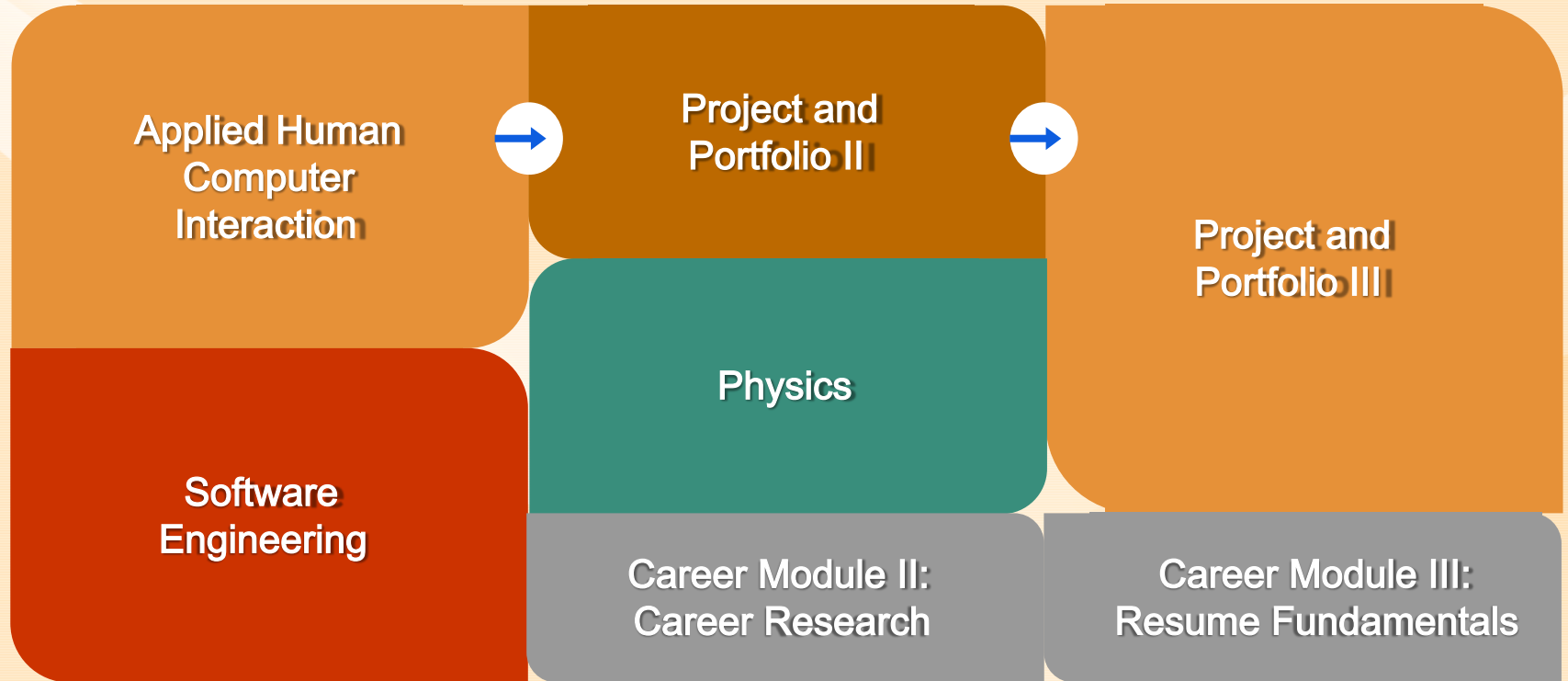
CD of PP3:

[joleske@fullsail.com](mailto:joleske@fullsail.com)

JohnOLeskeFS#4268

# Full Midterm Project Process

---



# Full Midterm Project Process

---

## Applied Human Computer Interaction

- Pre-Production
  - Design Document
  - Product Backlog
- AHI Topics
  - Nielsen's heuristics
  - Usability
  - UX
  - UI
  - Emerging tech

## Project and Portfolio II

- Core Functionality
  - Critical game systems
  - Interface and UI creation
- First Use/Playable
  - Playable complete
  - Experience
  - Fun factor

## Project and Portfolio III

- Alpha
  - Full Functionality
- Beta
  - Content complete
  - Balancing
- Finalizing
  - QA process
  - Presentation

---

# Project Expectations

---



# Game Expectations - Scope

- Medium/Indy scoped game
- Game similar to game from NES, SNES/Genesis era, or mobile and web platforms game tend to work best
- Focus on functionality over assets



# Game Expectations - Minimums

---

- All games must have at least 15 minutes of engaging and varied play
  - Most have far more than 15
- Must contain at least one single player mode
  - So the game can be easily demoed



# Game Expectations - Platform Support

---

- Must support a secondary platform
  - WebGL version exported to a webpage or published to a web portal
  - or
  - Playable on an android device (tablet or phone)
- Keep platforms in mind when making design and production choices
- Keep file size low
  - < 1GB
  - Under 512 MB preferred

# Game Expectations - Buildable

---

Design a game with your capabilities in mind

- Heavily story driven game
  - Someone on the team should be a writer
- Aesthetic as a hook (unique art styles)
  - Someone on the team should be able to source those assets
- 3D Animation heavy game
  - Someone on the team should be able to animate
- Game with 50 unique levels
  - Someone on the team should have the skills of a level designer

# Game Expectations - Expo

---

- Games will be presented at the FPS Expo
- First Thursday of the month after PP3
  - 11am-1pm



---

# Developer Expectations

---

# Expectations: Problem Solving

---

Put less importance on knowing things ahead of time.

- The job IS problem solving
  - On the job, you learn things just-in-time.
  - You must be able to figure out solutions on your own.
  - Unity Documentation is surprisingly good

# Expectations: Communication

---

Communication will be a challenge

- **Mandatory schedule**
  - CDs will see you 1 or 2 times a week
  - That isn't enough to keep communication open
- **Stay connected with us**
  - If something breaks, tell us
  - If there are issues with finding art, tell us
  - If something awesome changes in the project, tell us



# Expectations: Teamwork

---

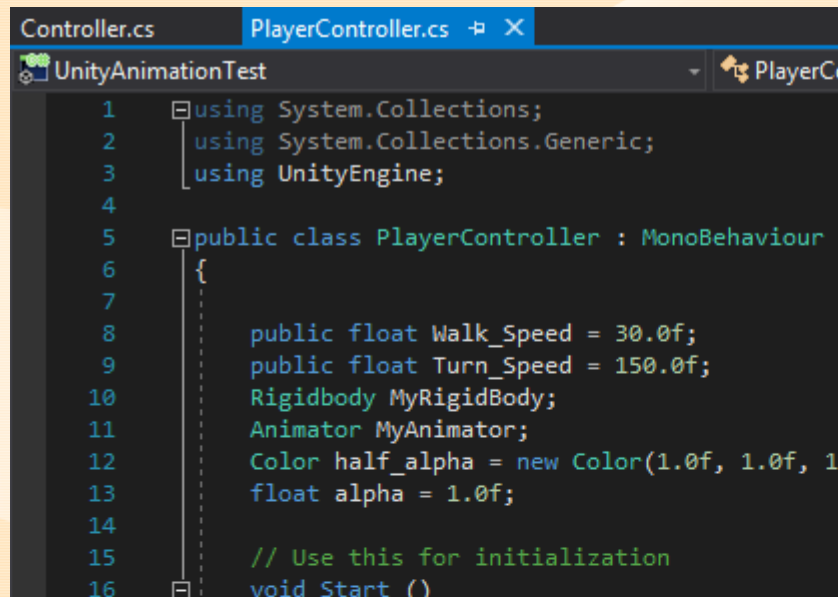
Work with each other

- Have a set schedule
  - Check in daily
  - Set a schedule for everyone to work TOGETHER
  - You will always get more done as a group

# Expectations: Be a developer

We will be doing a lot of different jobs

- Programmer
  - Creating functionality
  - Researching the engine
  - Fixing bugs



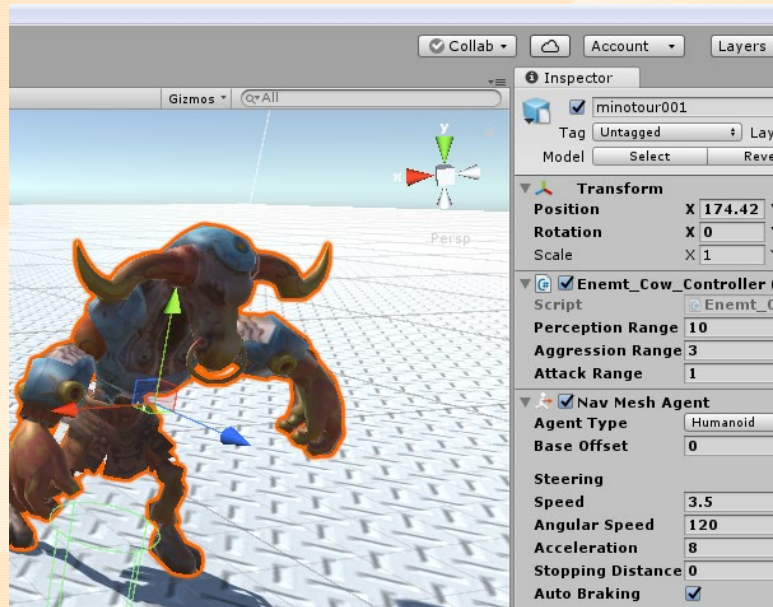
The screenshot shows a code editor with two tabs: 'Controller.cs' and 'PlayerController.cs'. The 'PlayerController.cs' tab is active. The code is written in C# and defines a 'PlayerController' class that inherits from 'MonoBehaviour'. The class contains several public float variables for 'Walk\_Speed' and 'Turn\_Speed', and private variables for 'Rigidbody', 'Animator', and 'Color'. It also includes a 'Start' method for initialization.

```
Controller.cs  PlayerController.cs  X
UnityAnimationTest
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PlayerController : MonoBehaviour
6  {
7
8      public float Walk_Speed = 30.0f;
9      public float Turn_Speed = 150.0f;
10     Rigidbody MyRigidbody;
11     Animator MyAnimator;
12     Color half_alpha = new Color(1.0f, 1.0f, 1.0f, 1.0f);
13     float alpha = 1.0f;
14
15     // Use this for initialization
16     void Start ()
```

# Expectations: Be a developer

We will be doing a lot of different jobs


- Designer
  - Defining mechanics
  - Level designs
  - Scripting Interactions
  - Balancing



# Expectations: Be a developer

We will be doing a lot of different jobs

- Producer
  - Defining expectations
  - Writing and following a Scheduling

 **Timed Spikes**  
in list [World Obstacles](#)  
☐ Recurring    Add #tags ▼    S/E & More ▼  
Labels  
**Difficulty: Easy** +  
Description  
[Edit](#)  
Intent:

- Spikes that pop into and out of the ground based on a timer that damage characters that collide with it.

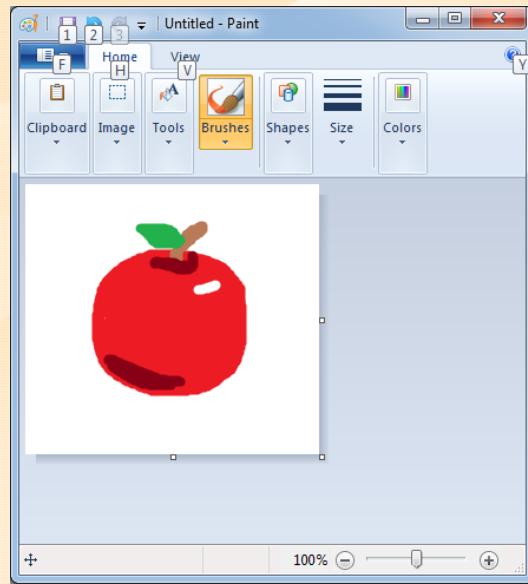
☒ **Test Cases / Acceptance Criteria** [Delete...](#)  
0%  
☐ Do spikes damage the player when the collide with the top of them?  
☐ Do spike not damage the player if the touch only the side of them?  
☐ Do the spikes go into and out of the ground based on a timer?  
☐ Do spikes not damage while in the ground?

# Expectations: Be a developer

We will be doing a lot of different jobs

- Artist

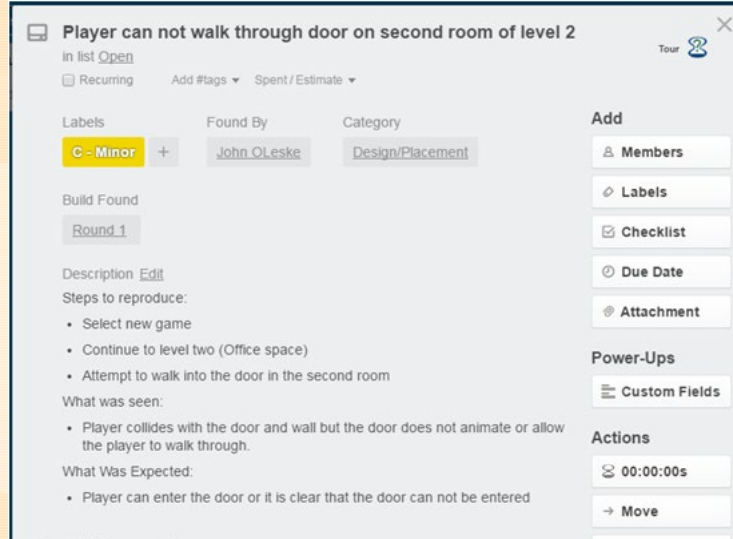
- Adding assets to product
- Setting up animations
- Creating simple assets (programmer art, placeholders)



# Expectations: Be a developer

We will be doing a lot of different jobs

- Quality Assurance Tester
  - Testing and reporting bug



The screenshot shows a bug report form with the following details:

- Title:** Player can not walk through door on second room of level 2
- Buttons:** In list [Open](#), ☐ Recurring, Add #tags, Spent / Estimate
- Labels:** C - Minor
- Found By:** John Oleske
- Category:** Design/Placement
- Build Found:** Round 1
- Description:** [Edit](#)
- Steps to reproduce:**
  - Select new game
  - Continue to level two (Office space)
  - Attempt to walk into the door in the second room
- What was seen:**
  - Player collides with the door and wall but the door does not animate or allow the player to walk through.
- What Was Expected:**
  - Player can enter the door or it is clear that the door can not be entered
- Right Sidebar:**
  - Add:** Members, Labels, Checklist, Due Date, Attachment
  - Power-Ups:** Custom Fields
  - Actions:** 00:00:00s, Move



---

# Project Policies

---

# Academic honesty

---

"Projects/Assignments: Students are expected to be honest and produce their own projects/assignments according to the specifications of their Course Director. **They must work solely on their projects/assignments unless otherwise noted by this Course Director.** Work submitted by our students is assumed to be a student's own thoughts, idea, and words. Discovery of the contrary will result in immediate consequences. **For group projects, all students whose names are submitted with the project are responsible for the content and will be subject to disciplinary action should plagiarism be discovered.**"

- Student Manual, page 17

# Academic honesty: Midterm Specifics

---

- Code/Functionality
- All functionality in the final product must be created by a student team member
- Any functionality not included in the unity installation must be authored by a student team member
  - Scripts
  - Prefabs
  - Scenes
  - If it can be made in unity you are expected to make it
- You may not use the unity asset store to add functionality to the project

# Academic honesty: Midterm Specifics

---

- Assets
- Assets authored by non-student team members may be used as long as there are legal rights to use the assets
  - Textures/sprites
  - Audio/sfx/music
  - Models/meshes
  - Animations
- Any assets used that was not created by a student team member must be have their source credited in the game's credits

# Academic honesty: Levels

---

- **Level 1** (0 score on the assignment and month's professionalism, conduct probation, and suspension):
  - Directly copying work from another source and submitting it as one's own.
  - Submitting work completed by another individual or student as one's own.
  - ...
- **Level 2** (0 score on the assignment and month's professionalism, and conduct probation):
  - Completing any work for another student that fulfills an academic requirement.
  - Knowingly furnishing false information to an instructor or any other representative of the University.
  - Repeated violations that fall under the Levels 3 or 4 headings.
- **Level 3** (0 score on the assignment and month's professionalism):
  - Submitting work that was turned in from another course or a previous attempt at this course without prior approval from the course instructor.
  - Significant omission or misuse of citation and/or references in course work.
  - In group work, including one's name to "tag along" on work of other team members in which he/she did not significantly contribute.
- **Level 4** (0 score on the month's professionalism):
  - In group work, allowing a team member to include his/her name to "tag along" on the work of other team members despite having not significantly contributed to that work.

# Academic honesty

---

If you are unsure, ask

- There can be some grey areas with this as time goes on.
- If an item is in question, bring it up to staff



---

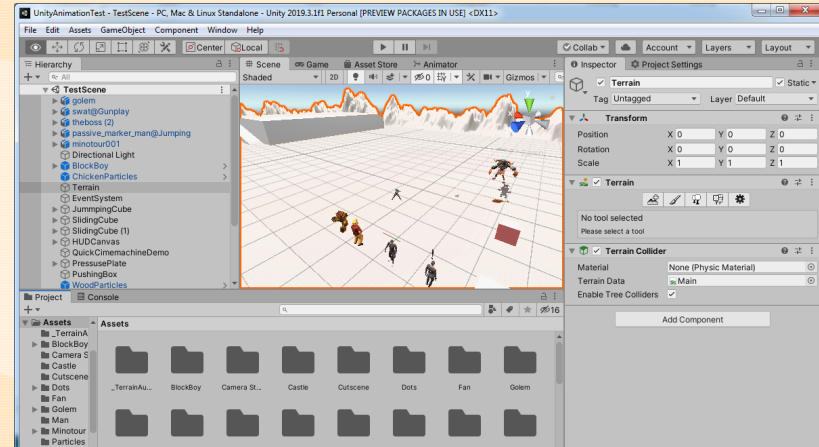
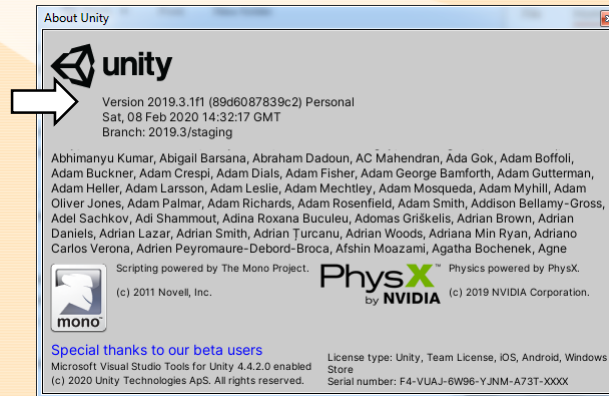
# Tools

---

# Unity3D

Unity3d.com

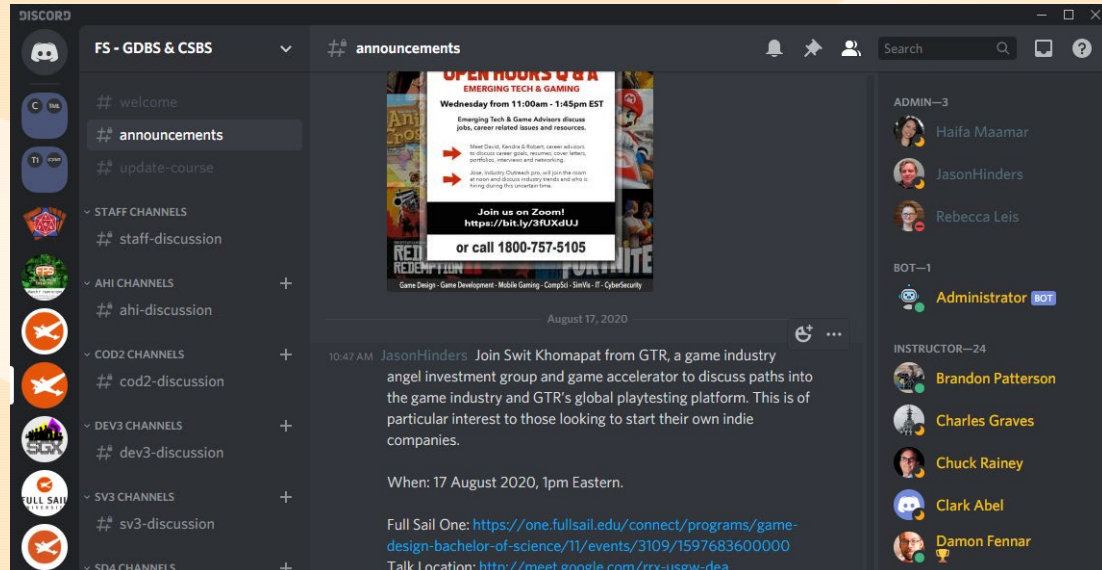
- Our dev environment
- Scripting in c#
- Ensure each team member is using the same version



# Discord

discordapp.com

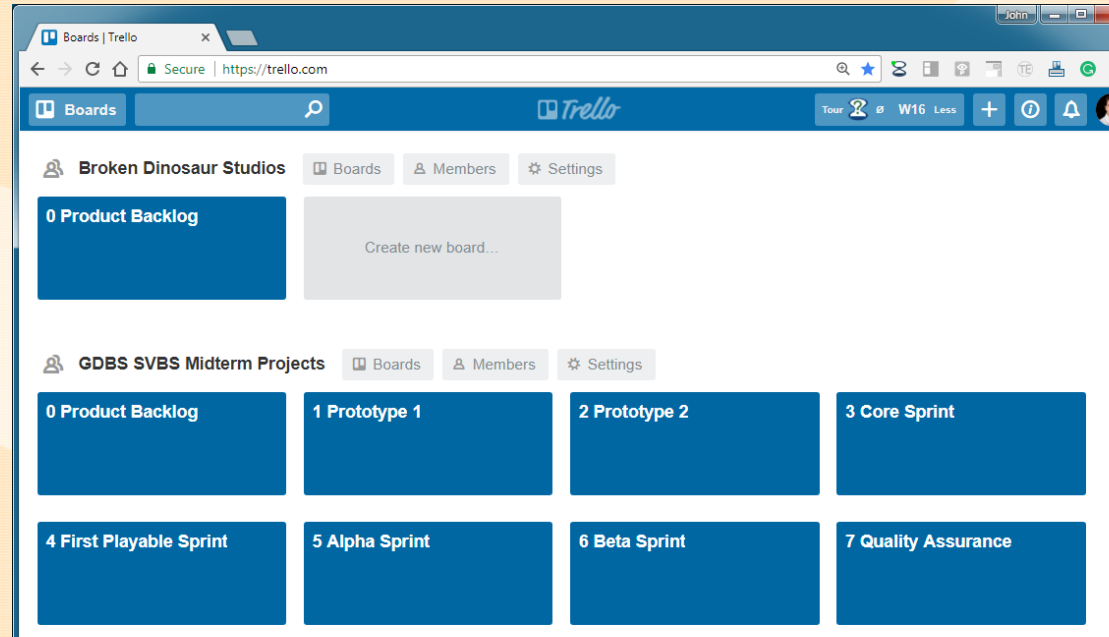
- Keep communication open while working remotely
- Share files that are not part of the project



# Trello

trello.com

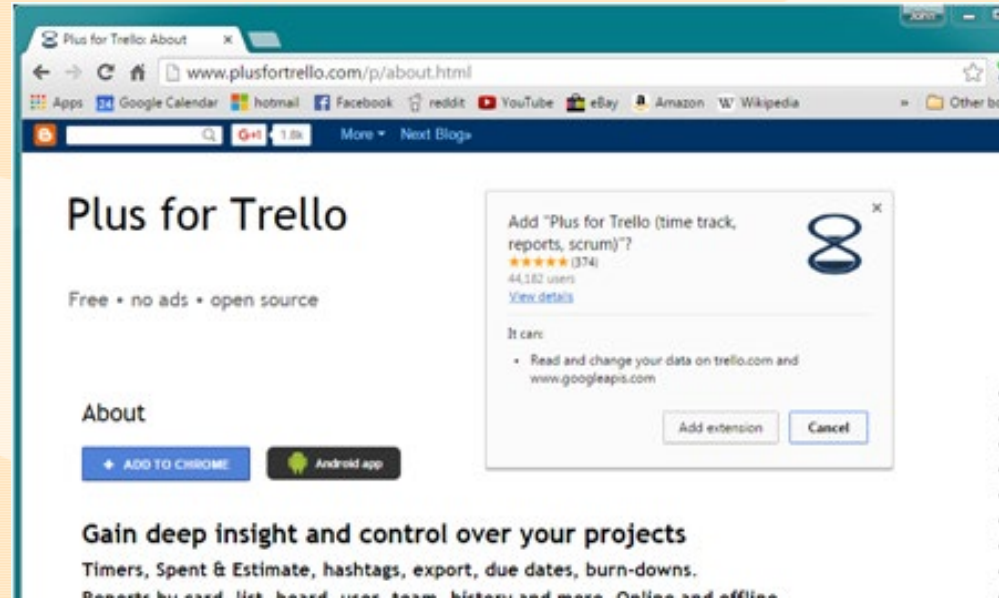
- Part of our design space
- Our task management system



# Plus for trello (Chrome plugin)

plusfortrello.com

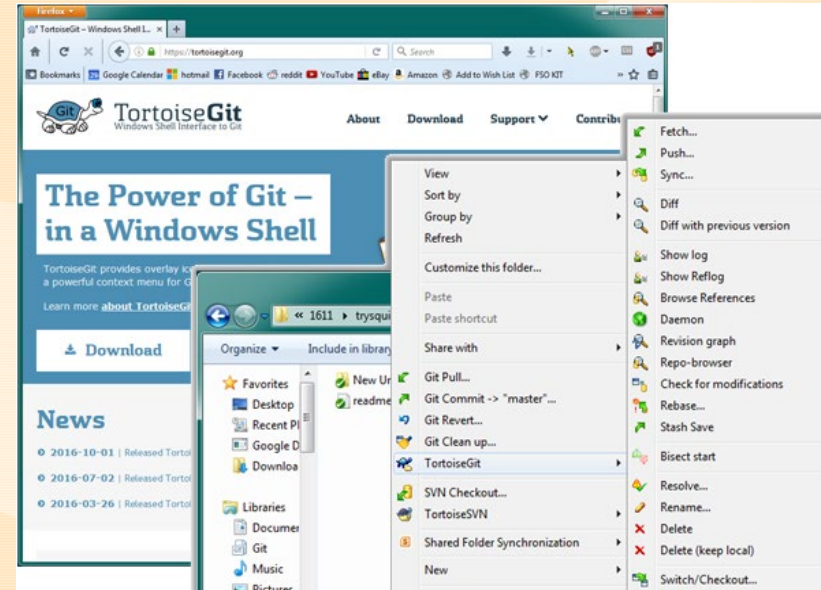
- Task ownership
- Task hour tracking



# TortoiseGit

tortoisegit.org

- Our Git versioning client

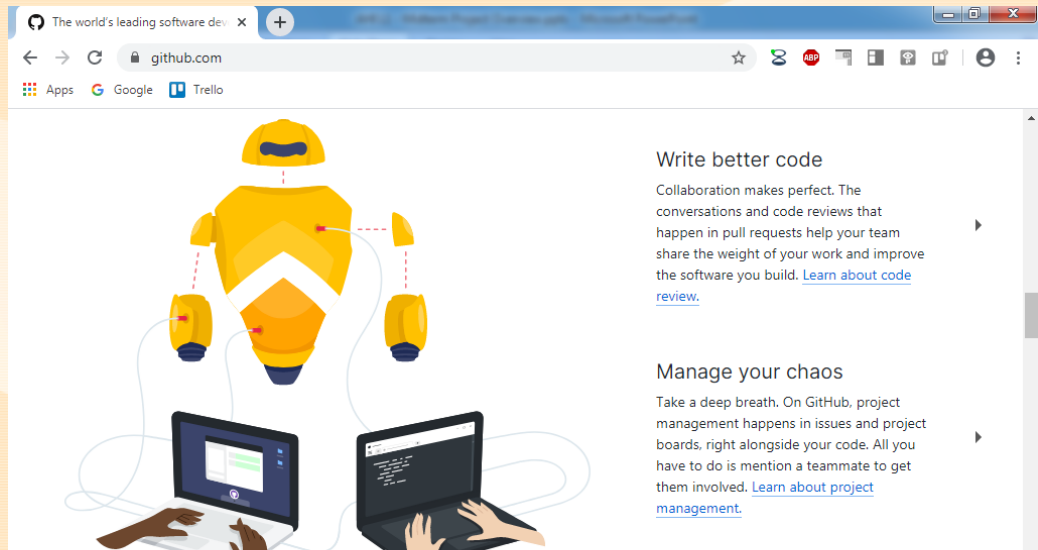




# Git Hub

github.com

- Git server will be provided through GitHub



# Git LFS

---

`git-lfs.github.com`

- Allows github to receive large sized files
- Larger than 50MB (github max file file)
- (Zipped milestones, Video files, Resource packages ...)



**Git Large File Storage**

**An open source Git extension for  
versioning large files**

Git Large File Storage (LFS) replaces large files such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server like GitHub.com or GitHub Enterprise.

# Install, Create Accounts, or Confirm

---

- Unity  
(unity3d.com)  
Everyone on same version
- TortoiseGit  
(tortoisegit.org)
- Git framework  
(git-scm.com)
- Git LFS  
(git-lfs.github.com)
- Discord  
(discordapp.com)
- Trello  
(trello.com)  
Accounts created
- Chrome  
(google.com/chrome)
- Plus for trello plugin  
(plusfortrello.com)  
Sync method set to  
“Recommended- Store inside  
Trello (S/E Trello card  
comments)”
- Github  
Accounts confirmed

# <Activity> Form Teams

---

## Form team

- 3-5 students per team
- Will be working with each other for months

## Collect the info

- Team
  - Team name
  - Discord channel
- For each team member
  - Full name
  - Trello username
    - (The one in the parenthesis next to your full name on the webpage)
  - Github username
  - Discord username
  - Email

# <Activity> Discuss game ideas

---

I will be setting up servers and documents for all of the teams

Think of a game

- Discuss as a team what kind of game you would like to develop

# <Activity> Discuss game ideas

---

Form a basic game elevator pitch for your game

- The game's elevator pitch.
  - A one or two sentence description of the game.
  - This should encapsulate the hook of the game.
  - Why would someone want to buy and play this game over other games?
  - Why should a company be willing to invest in making this product?
- [Name] is [genre] in [setting] where player [plays as/action] to [win condition] using/experiencing [hook]

---

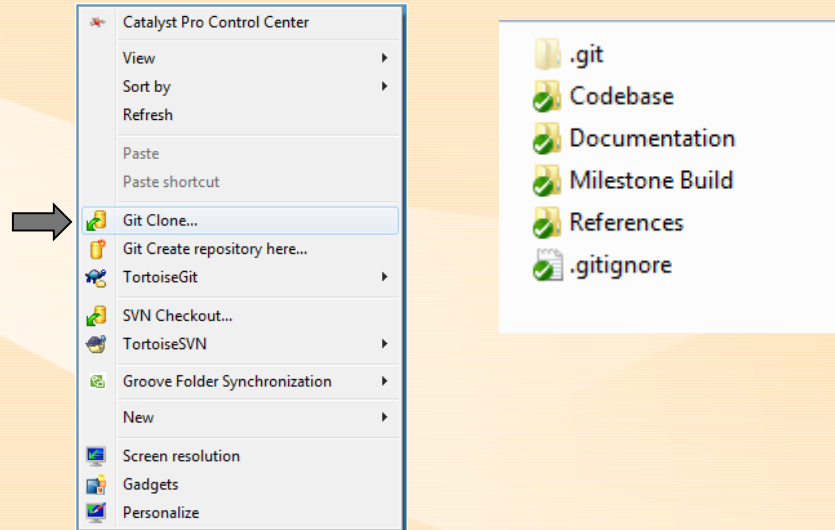
# Clone The Repo

---

# <Activity> Clone the project folder

Clone the repo

- After the clone the project folder should look like this

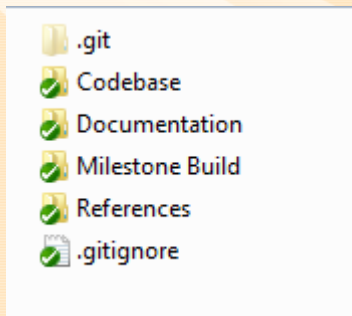




# <Activity> Let's explore the files

---

- **.Git**
  - The hidden folder containing the local repo
- **Codebase**
  - Your unity project
- **Documentation**
  - Preproduction and production documents to be filled out
- **Milestone Build**
  - A single most recent good build of the game
- **References**
  - Lecture power points, rubrics, FAQs, Project archives
- **.gitignore**
  - The file that defines what should not be pushed to the repo



---

# Pre-Production

---

# The purpose of Preproduction

---

- Pre-production
  - Make decisions on the product
  - Prove the validity of those decisions with rapid iteration of prototypes
  - Throw out what doesn't work and keep what does
  - Document the full scope based on the above
- Production
  - Make the rest of game based on the above

# The purpose of Preproduction

---

- As hard as preproduction is, production is much harder without it
  - And more expensive in the long run
- Fixing problems on paper takes less time and resources than implementing the system to see it doesn't work.

# Top Down Versus Bottom up

---

- Top down
  - Starting at the big picture and breaking down into smaller and smaller components
  - What do we want? Now how can we make that happen?
- Bottom Up
  - Defining the components and integrating and combining to get to a bigger picture
  - What can we do? Now what can we make with that?
- Important to look at the game from both points of view

---

# Trello

Our task management platform

---

# Trello: Our task management platform

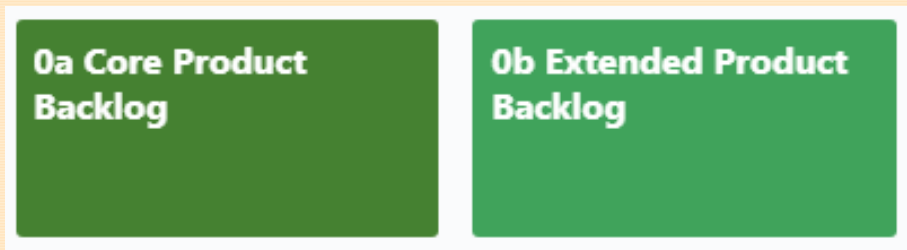
---

- Bottom up design
  - Defining the individual aspects we intend to create to get to a whole picture
- Brainstorm features
- Create new card on trello for each feature to be added to the game
  - Player action
  - Character
  - Enemy type
  - Item
  - Weapon
  - Power up
  - Game mode
  - ...

# Trello: Our task management platform

---

## Our team's Trello boards



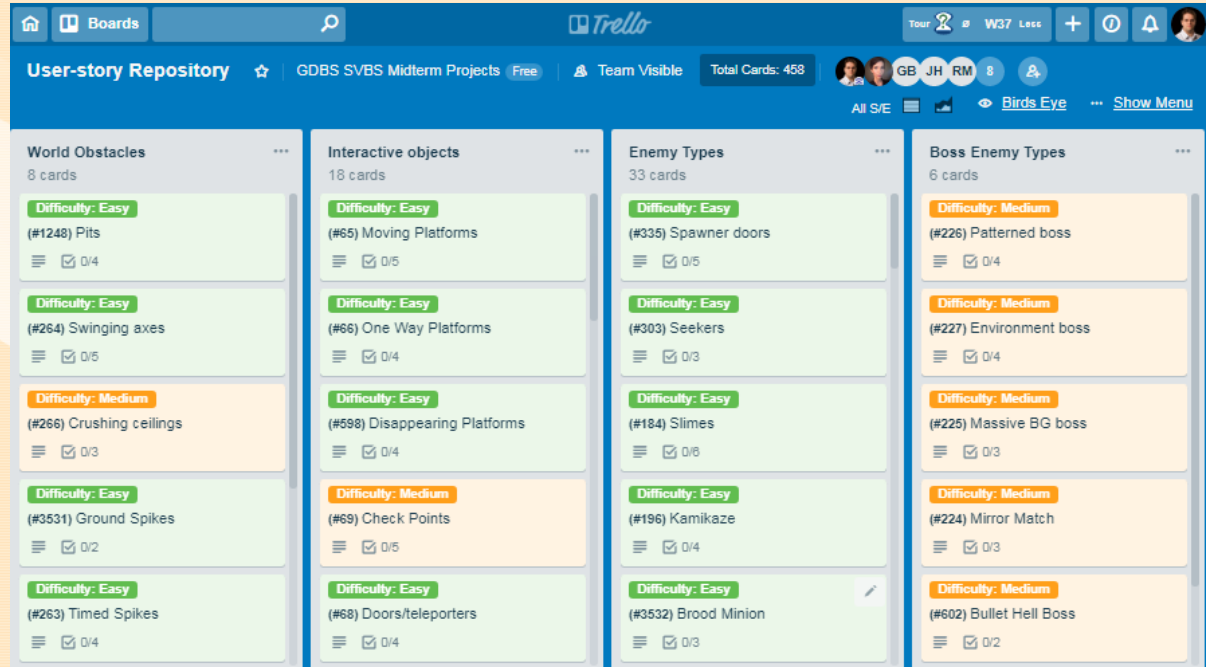
- “0a Core Product Backlog”
  - All items and features required to create the product's vertical slice
  - M.V.P. Minimum viable product
- “0b Extended Product Backlog”
  - All the wish list, would be cool to have, stretch goals... of the product



# Trello

By lecture 2 our trello boards should look something like this

- Shoot for 100-150 items/features across the 2 boards



---

# The Design Document

---

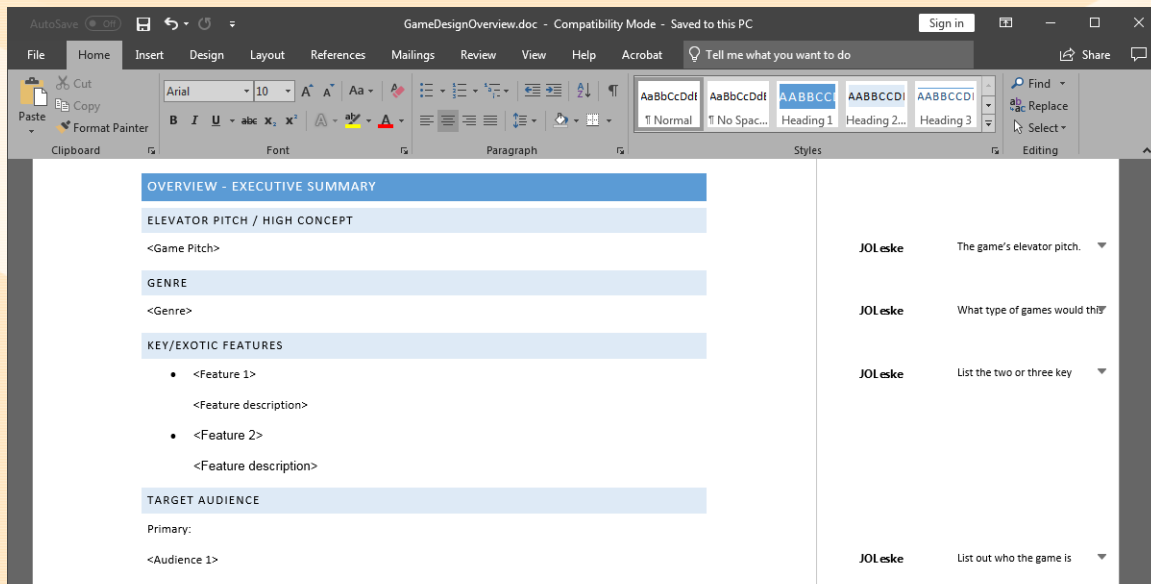
# Design Document

---

- Top down design
  - Defining the intent of the product as a whole and breaking that down into its aspects
- Agree on the core design
  - Overall goals
  - Fun Factor
  - Selling points of the game
  - Overarching systems and mechanics

# Design Document

- Template document has been provided for you
  - In your repository's documentation folder
  - This absolutely must be customized for the game you are making
    - All games are different
    - Must describe the aspects of the game you are making



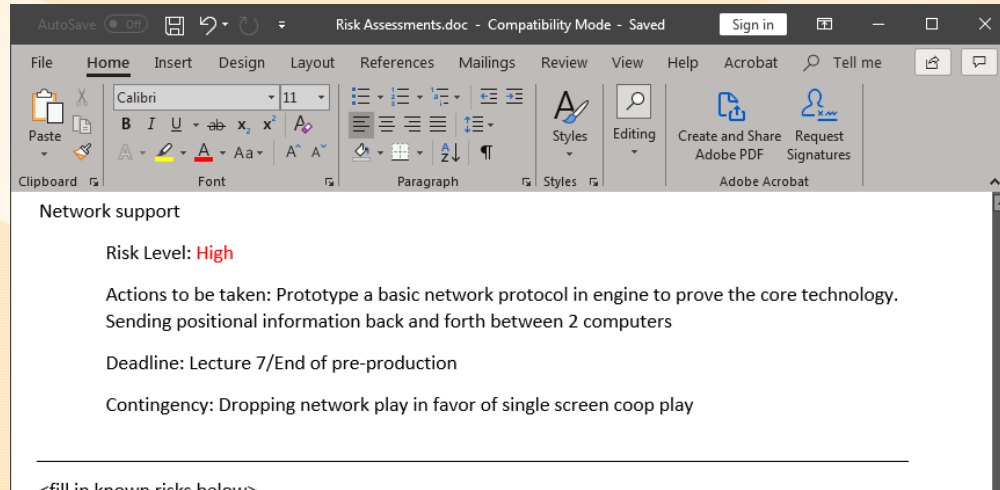
---

# Risk Assessments

---

# Risk Assessments

- All known risks identified on the project
  - How dangerous the risk is to the overall project plan
  - What actions must be taken to mitigate the risk
  - Deadline for those actions
  - Contingency plan
- Template document has been provided for you.
  - In your repository's documentation folder



---

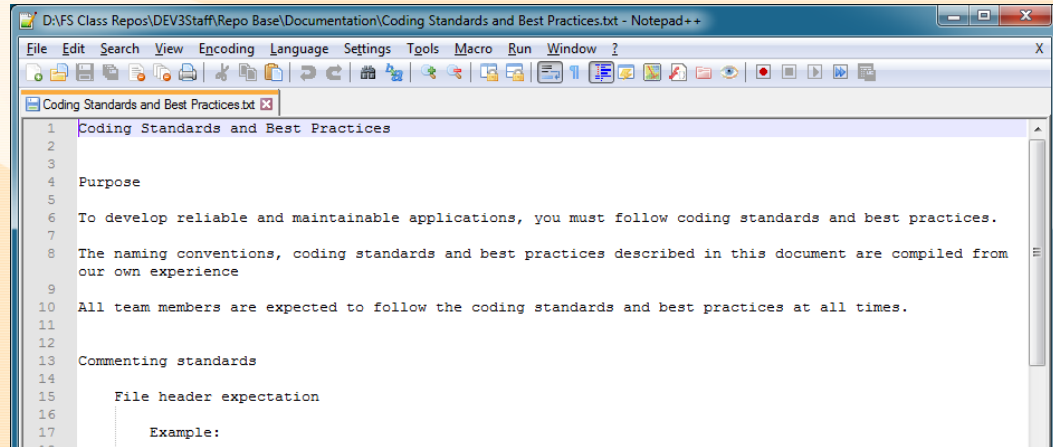
# Coding Standards and Best Practices

---

# Coding Standards and Best Practices

The team agreed to a standard

- Commenting
  - Naming conventions
  - File Formatting
  - Indentation and Spacing
  - General Programming practices
- Template document has been provided for you.
    - In your repository's documentation folder





---

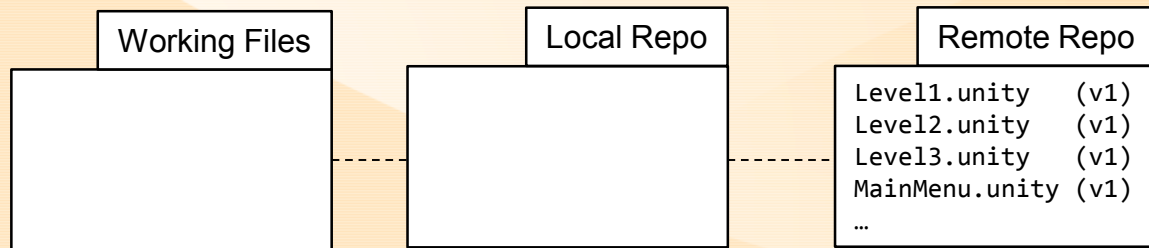
# Version Control

---

# Git Basics

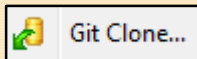
---

- Understand the system behind the interface
  - Three main sections to pay attention to
    - Working files
    - Local Repository
    - Remote repository
  - (actually 4 with the staging phase but that mostly handles itself)

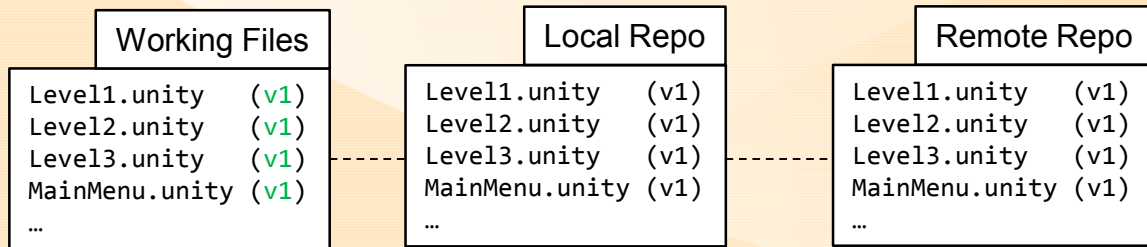


# Clone

Contributors need to clone the repository to start working on the shared files.

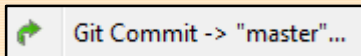


- Get a bring remote repository into the local repository
- Populate working files from local repository

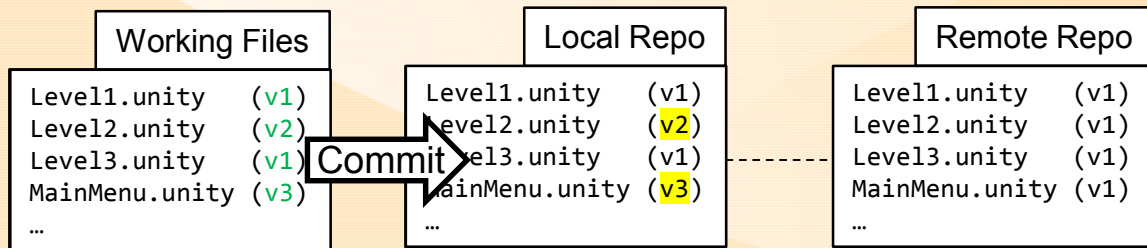


# Commit

Once changes to the files has been made that work need to be committed

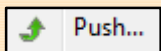


- Commit saves the changes to the local repository
- Once committed there is a timestamp of the files that can always be returned to

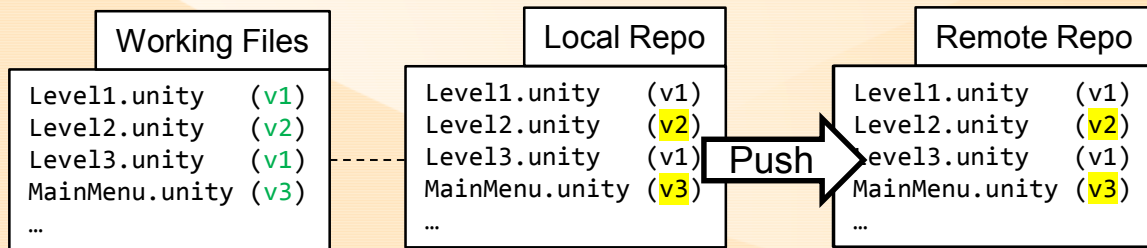


# Push

Push integrate changes onto the remote repository



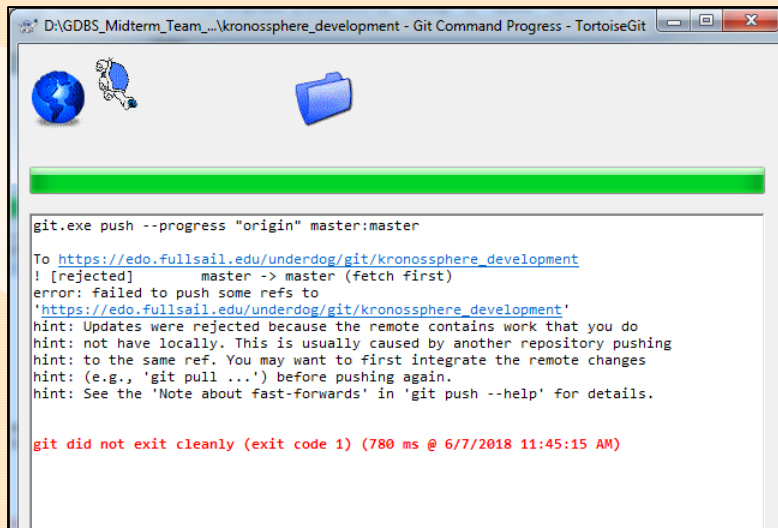
- Non committed work does not get pushed
- If a file hasn't been added it doesn't get committed



# Push: Changes on remote error

“error: failed to push some refs to...  
hint: updates were rejected because the  
remote contains work that you do not have  
locally”

(Read all of the error message, not just the  
red text)

A screenshot of a TortoiseGit window titled "D:\GDBS\_Midterm\_Team\_... \kronosphere\_development - Git Command Progress - TortoiseGit". The window has a green progress bar at the top. Below it, the command "git.exe push --progress "origin" master:master" is shown. The output text indicates a rejection of the push because the remote contains work not present locally. It includes a URL to the repository and several hints for resolving the issue, such as pulling first. At the bottom, a red line of text states "git did not exit cleanly (exit code 1) (780 ms @ 6/7/2018 11:45:15 AM)".

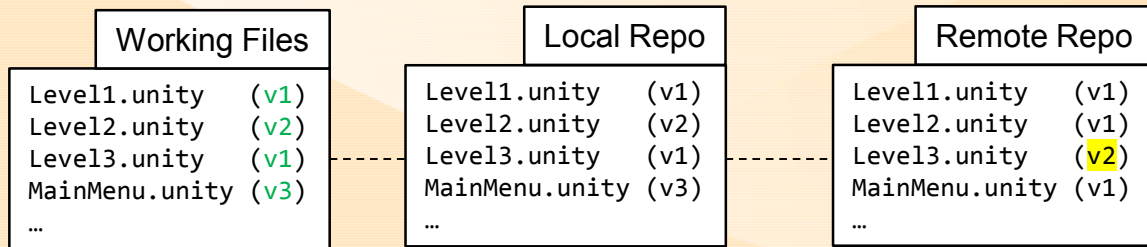
```
git.exe push --progress "origin" master:master

To https://edo.fullsail.edu/underdog/git/kronosphere_development
! [rejected]        master -> master (fetch first)
error: failed to push some refs to
'https://edo.fullsail.edu/underdog/git/kronosphere_development'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

git did not exit cleanly (exit code 1) (780 ms @ 6/7/2018 11:45:15 AM)
```

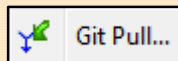
# Push

You can't push if there are changes on the remote server that you do not have on your local repo

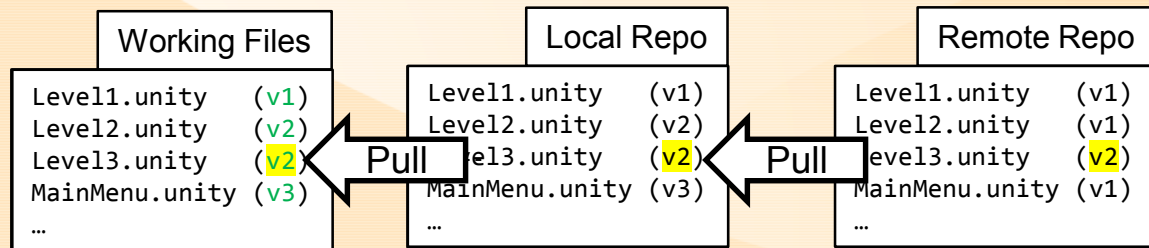


# Pull

Pull changes from the remote and integrate them onto your build



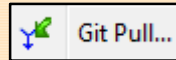
- Changes get integrated into the local repo
- If integration is good the working files change to match





# Pull

Do not pull with unity open

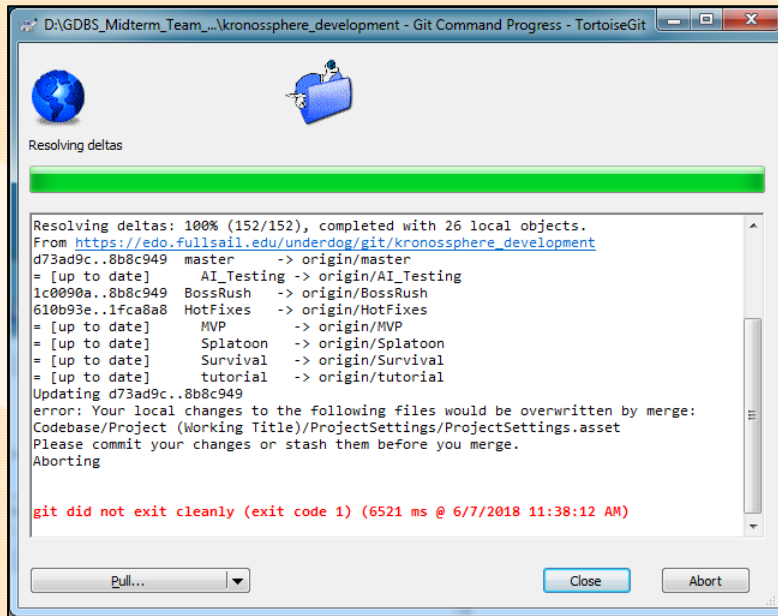


- Unity is always updating and recompiling based on changes in the files
- If you pull with unity open unity and there is an error unity will attempt to recompile with the error and break



# Pull: Uncommitted work error

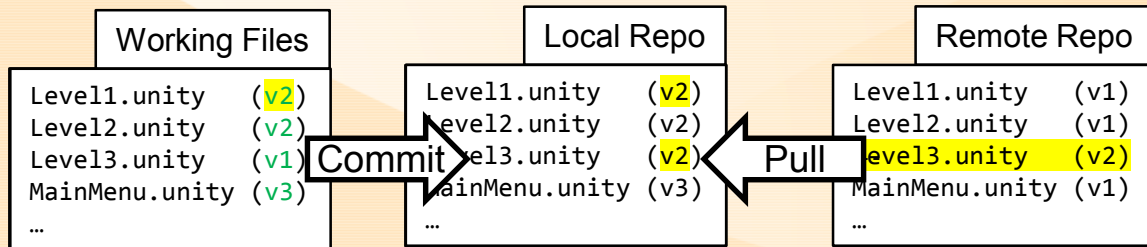
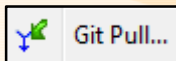
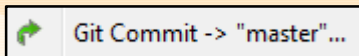
“error: Your local changes to the following files would be overwritten by merge... please commit your changes or stash them before you merge.”



# Uncommitted work error

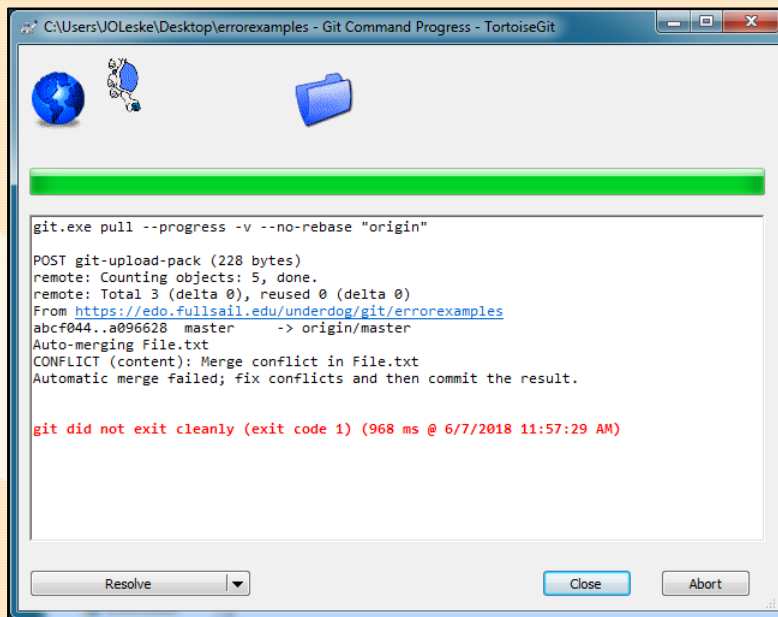
## Resolution

- You have changes to your working files that have not been committed
- Once you commit your changes you can pull and get the remote changes



# Conflict

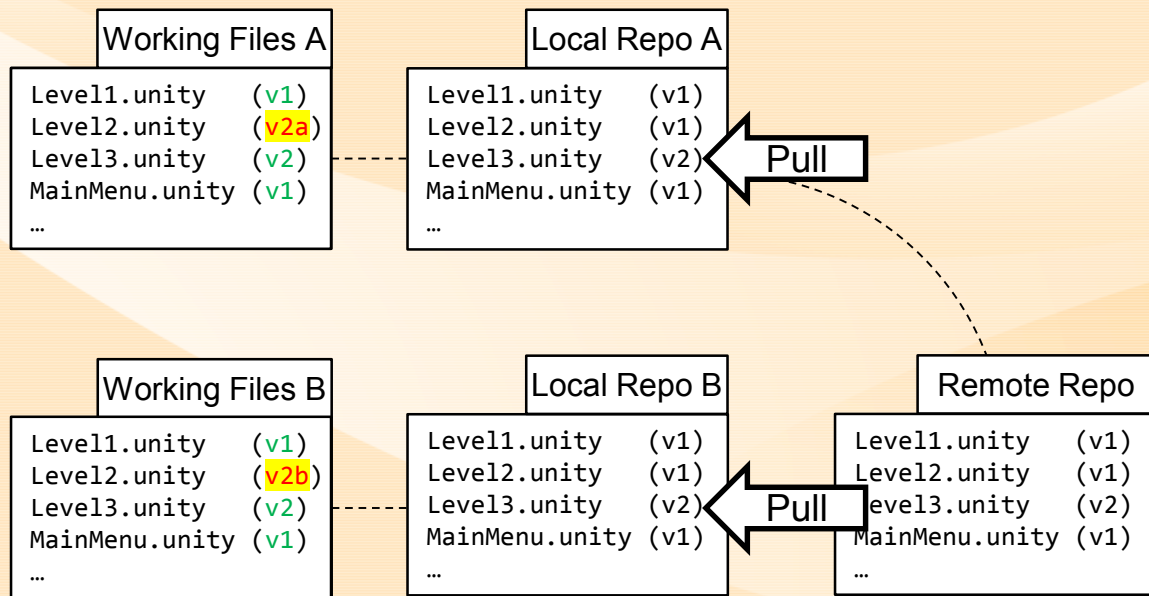
“Automatic merge failed: fix conflicts and then commit the results”



# Conflict

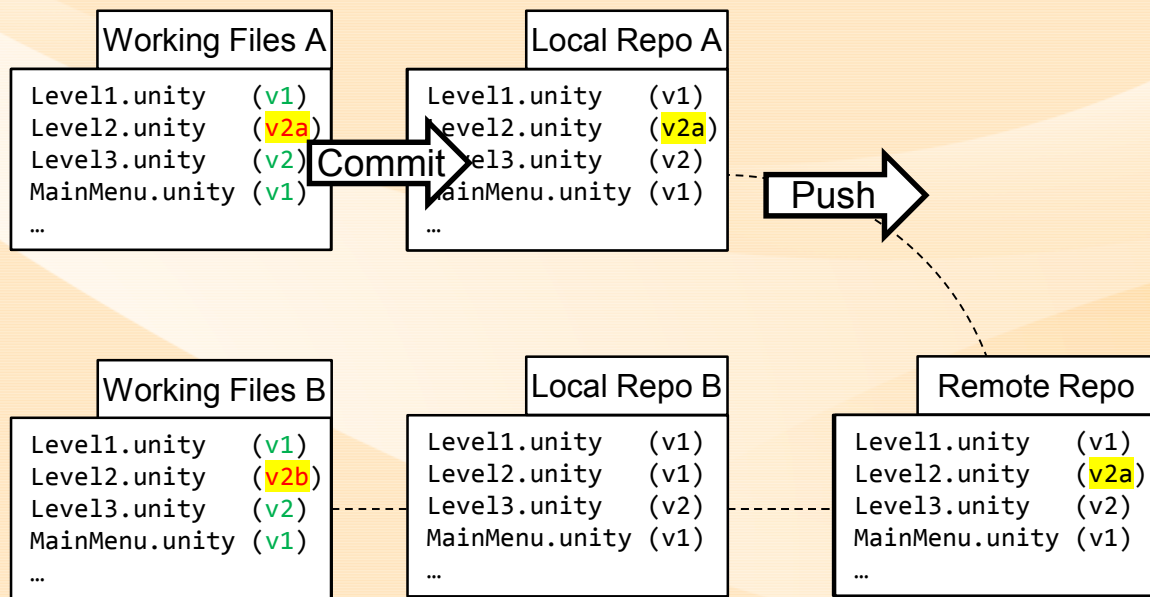
Conflicts are created when 2 pull and then modify the same file

These two Devs already have a conflict though they don't necessarily know it



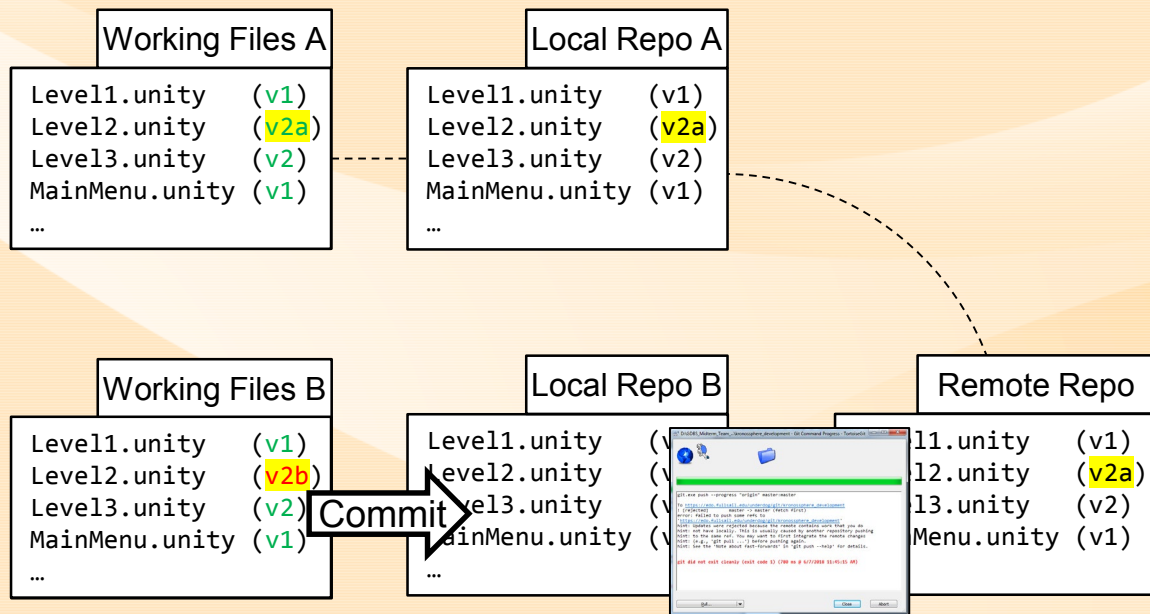
# Conflict

The first dev will be able to commit and push with no error



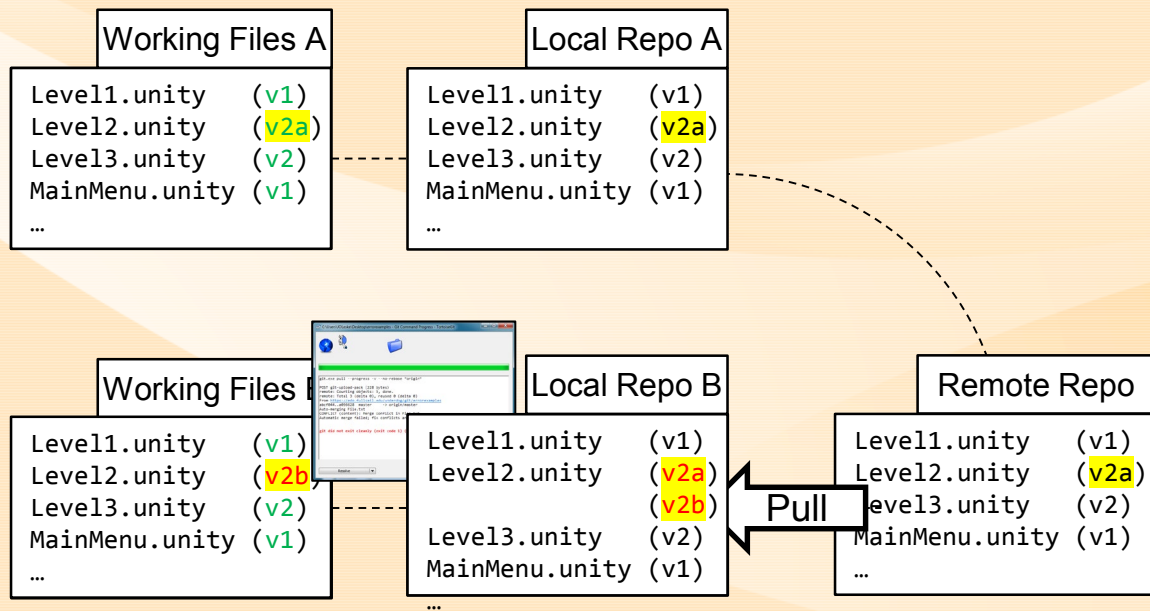
# Conflict

The second dev will be able to commit but will be blocked by the “remote contains work that you do not have” error



# Conflict

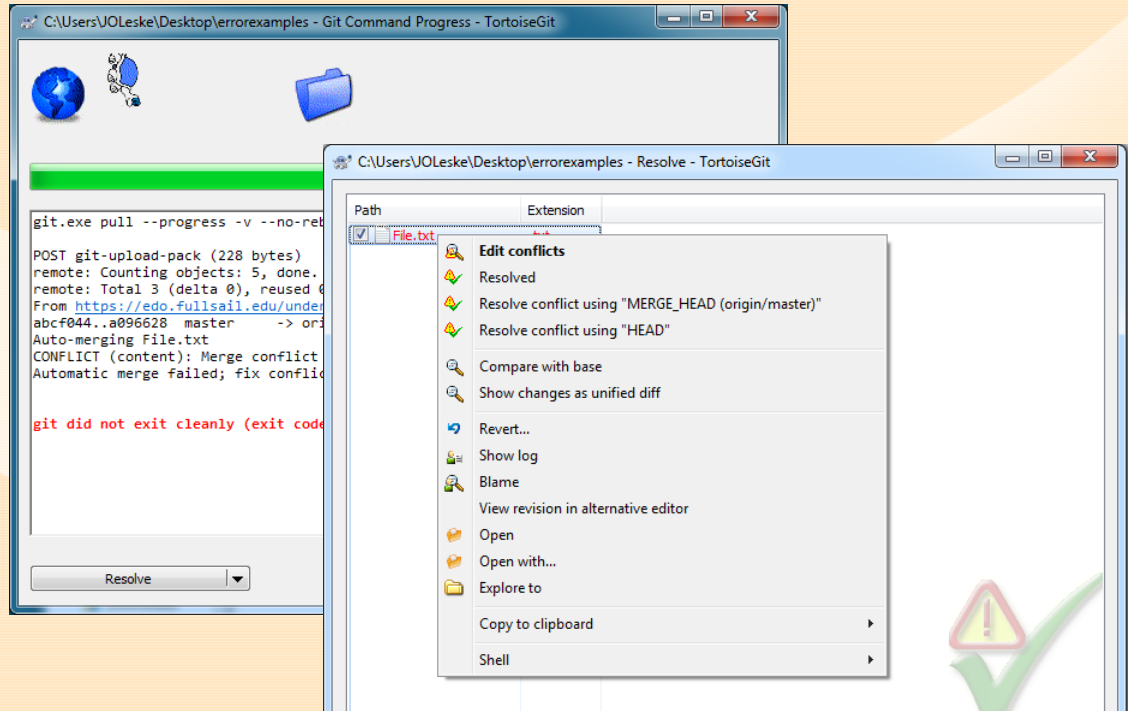
When the second dev pulls to fix the first error they will then get the “conflict” error because git doesn't know what to do with the changes from DevA and DevB since the both changes the same file





# Conflict

There are 4 main options to resolve a conflicted file

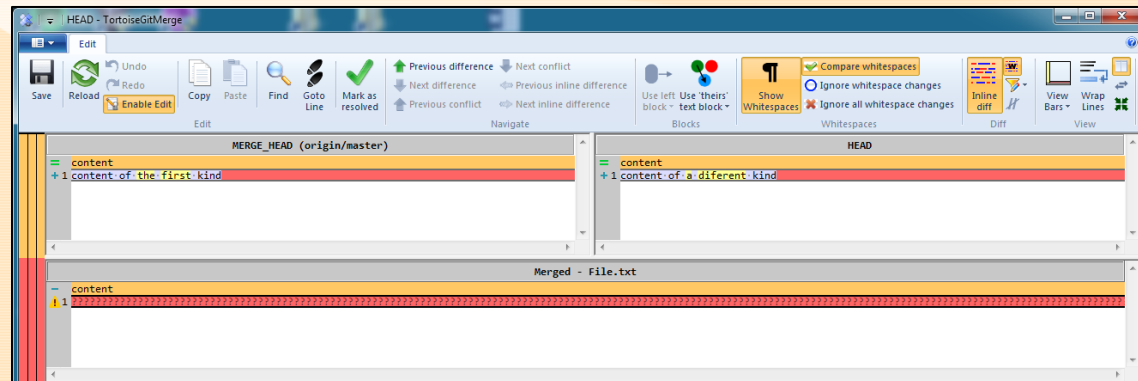


# Conflict: Resolution

## Edit Conflicts



- A merge tool can be used to pick and choose between lines and choose what in each should be saved
- Only works on text based files
  - Code
  - Text based assets

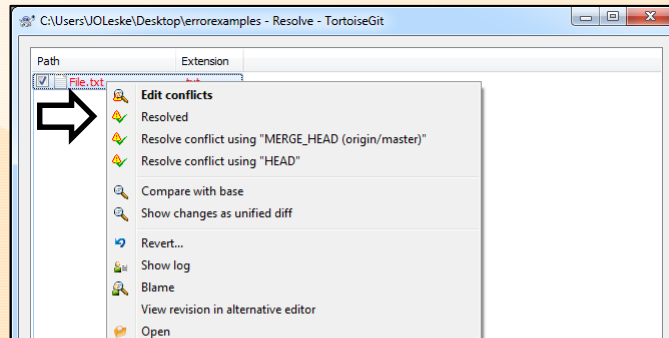


# Conflict: Resolution

Resolved




- If the conflict is resolved manually without using the git interface they can simply be marked as resolved
- DO NOT do this unless you have actually fixed the conflict elsewhere

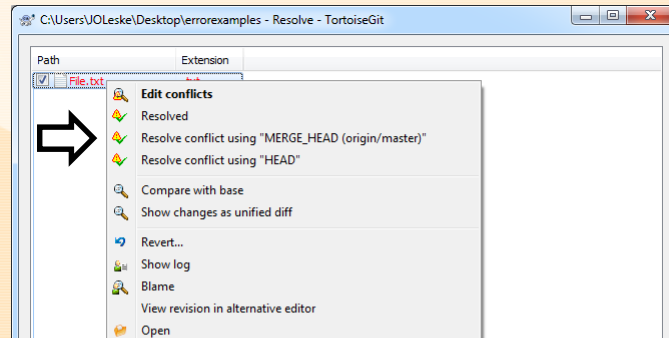


# Conflict: Resolution

Resolve using “MERGE HEAD”

 Resolve conflict using "MERGE\_HEAD (origin/master)"

- Take all of my changes and throw them away to keep all the changes from the remote repo

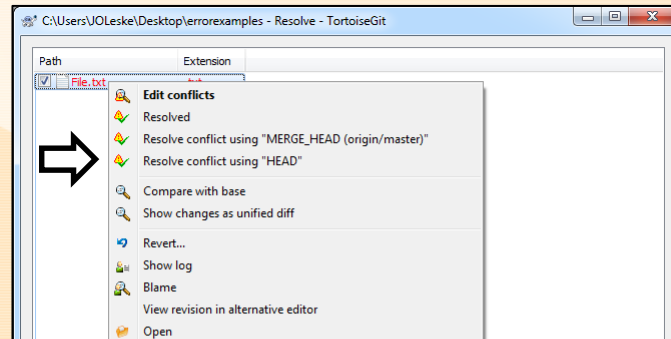


# Conflict: Resolution

Resolve using “HEAD”

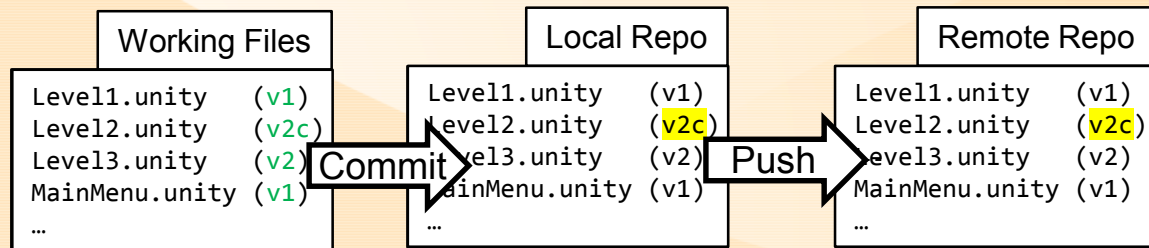
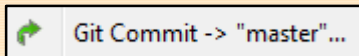


- Take all of the changes from the remote repo and throw them away to keep all of mine



# Commit

After conflicts have been resolved you have to commit those changes



---

# Git

## Tips and best practices

---

# Tips: Avoid having to merge

---

- Don't work on the same files at the same time
  - Binary files (non text) cannot be merged with Git
  - Don't work in the same scene at the same time
  - Don't work on the same prefabs at the same time
- Communicate and check in and check out assets that everyone contributes to
  - Scene files and Prefab files being the most common sources of conflicts



# Tips: Use Sandbox Scenes

---

- Organize the shared scenes
  - Scene for each level
- Use sandboxes
  - Each person on the team needs their own work scene.
  - Complete as much work as you can and test in the sandbox
- Integrate into shared scenes
  - Only integrate after it has been tested to work on the sandbox scenes
  - Need to be sure only one person works on a scene at a time

# Tips: Integrate Often

---

- Integrate to the remote server each time a task is complete
- The more time between pulls the more merge problems you can encounter

# Tips: Do not bloat the server space

---

- Time is a resource
  - Don't push giant packages of resources if you are only using 1 thing from it
  - Huge repos take longer to push and pull
- Can cause down time when things go wrong
  - When the server is out of space no one can push or pull until the server space is freed and reset manually

# Tips: Only use 1 versioning system

---

- Don't use git in a folder that is synced with another application
  - One drive
  - Google drive
  - Dropbox
  - ...
- Using 2 versioning systems at the same time can cause unexpected and unfixable issues

# Tips: Learn from errors

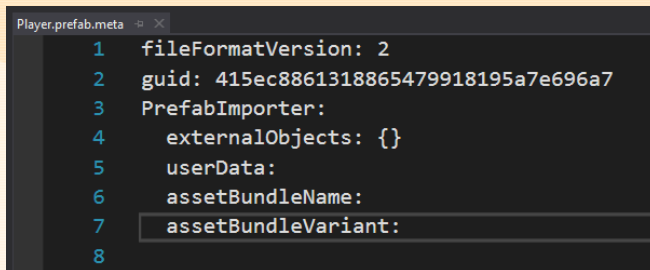
---

- If things go wrong learn why and fix the problem in your process
- TortoiseGit does give useful error messages but you have to read the whole thing
  - Not just the red text
  - There is also the error help doc and contacting me on discord in case you get stuck
- Git does work
  - Most used version control system at present
  - Don't blame the hammer when you hit your thumb

# Unity: Meta files

---

- How they work
  - Unity generates meta files for all files in the assets directory
  - Meta files contain GUIDs
  - Every reference to that asset/object in unity is done using that GUID as a reference
  - Deleting or regenerated a meta files makes all objects lose their references



A screenshot of a code editor window showing the contents of a Unity meta file named 'Player.prefab.meta'. The file is a JSON object with the following structure:

```
1  fileFormatVersion: 2
2  guid: 415ec8861318865479918195a7e696a7
3  PrefabImporter:
4    externalObjects: {}
5  userData:
6  assetBundleName:
7  assetBundleVariant:
```

- Don't delete them
- Don't regenerate them
- Once one is versioned keep that version

# Additional help if needed

---

- Common Git Errors
  - <https://bit.ly/2vXfkrd>
- Working with Unity and Git
  - <https://bit.ly/38HWy4g>
- Contact Staff
  - More than happy to help you resolve a git issues
- Lot and lots of additional online tutorials for git. Most working with command line interface.

---

# Activities

---



# <Activity> Git crash course

---

Let's use git on a team

- Create sandbox scenes in the scenes folder
  - Each team member create their own
- Coding Standards and Best Practices
  - Each team member add your name to the file
  - (text document)
- Design document
  - Each team member add your name to the file
  - (Binary file)
- Sync versions to get everyone's changes
- You will get conflicts and errors
  - That's the point of the activity

---

# Assignment

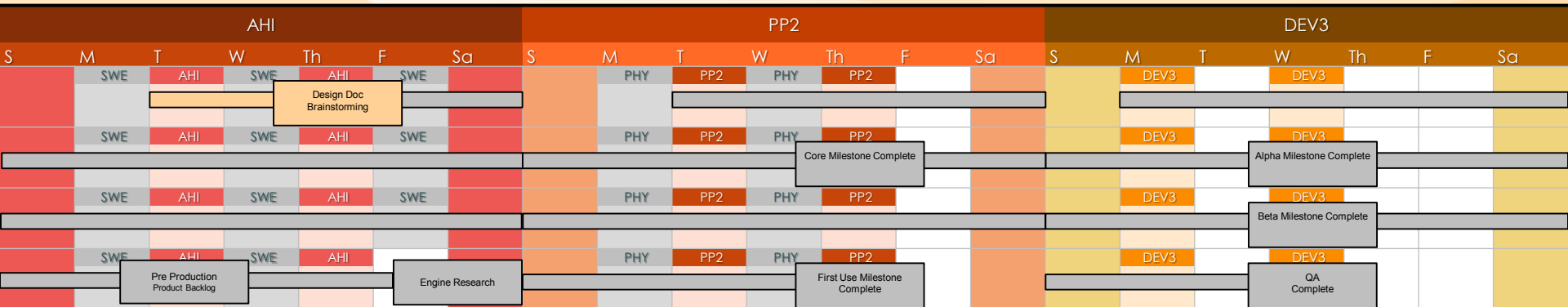
---

# Assignment

## By Lecture 2

- Pre pro
  - Trello boards filled out
    - Brainstorming features completed
  - Design Document worked on
  - Risk assessments worked on
  - Coding Standards and Best Practices agreed upon

(drafts ready for review)



# Contact Info

---

- John OLeske
  - JOLeske@fullsail.com
  - Discord: JohnOLeskeFS#4268
  - Skype:joleske
  - Trello: johnoleske
  - Work Phone: 407.551.2024 x 8926
  - Google: joleskefs@gmail.com
- Office hours
  - Mon 1-5 Friday 1-5
  - By request available
- Pre-Production Frequently Asked Questions
  - <https://bit.ly/2wNvmnA>