# Table of Contents

[Table of Contents 0](#_Toc504482030)

[Anxieteam 2](#_Toc504482031)

[Development Environment 2](#_Toc504482032)

[Unity3D 2](#_Toc504482033)

[Visual Studio 2](#_Toc504482034)

[Git 3](#_Toc504482035)

[File Structure 3](#_Toc504482036)

[Repository Information 3](#_Toc504482037)

[Photoshop 6](#_Toc504482038)

[Illustrator 6](#_Toc504482039)

[Redmine/Pineapple 6](#_Toc504482040)

[Skype 6](#_Toc504482041)

[Slack 6](#_Toc504482042)

[Delivery Platform 6](#_Toc504482043)

[Game Mechanics and Systems 8](#_Toc504482044)

[Movement 8](#_Toc504482045)

[Horizontal (L/R) 8](#_Toc504482046)

[Thrust (Forward) 8](#_Toc504482047)

[Levels 8](#_Toc504482048)

[Linear Map 8](#_Toc504482049)

[Score 8](#_Toc504482050)

[Enemies 9](#_Toc504482051)

[Collisions / Debris 9](#_Toc504482052)

[Shooting 9](#_Toc504482053)

[Win-States 9](#_Toc504482054)

[User Input 9](#_Toc504482055)

[Art Pipeline 10](#_Toc504482056)

[Concept to Digital 10](#_Toc504482057)

[Digital to Repository 10](#_Toc504482058)

[Repository to Unity 10](#_Toc504482059)

[Unity to Game 10](#_Toc504482060)

[Design Pipeline 11](#_Toc504482061)

[Unity Engine 11](#_Toc504482062)

[Milestone Updates 13](#_Toc504482063)

[Milestone 1A 13](#_Toc504482064)

[Milestone 4 14](#_Toc504482065)

[Milestone 5 15](#_Toc504482066)

# 4.5D Studios

## Development Environment

### Unity3D

Risk: Medium-Low

We are using Unity's 3D game engine to build our game on. In doing so, we are able to jump right into the game's development cycle without having to worry about building the engine itself. Unity does come with some inherent setbacks, such as being unable to modifier the main game loop, but these setbacks are inconsequential for the scale of our project. Unity is a good fit for the team as all have had experience working with Unity prior. It comes with preprogrammed physics, colliders, and an animation machine for a 2-dimenional environment. The programmer(s) are familiar with C#, and has proficient to extensive experience with C++ and other C languages. Unity also comes with in-depth documentation on how to use the engine.

Among Unity’s abilities is the ability to make prefabrications (prefabs) which allow the designer(s)/programmer(s) to easily make one instance of an object or unit, and duplicate it while maintaining the same settings between instances. Unity also provides access to public variables in custom scripts so that non-programming team members can test various settings for balancing the game. Unity also comes with a built-in UI system, allowing designer(s)/programmer(s) to dive right into Heads-Up-Display elements or menu screens without needing the programmer(s) to create a structure for it. When looking at publishing options, Unity provides multiple build settings to allow for publishing to multiple platforms, as well as any compatibility that needs to be built into a game to do so. One of the only drawbacks is that the artists have limited experience with Unity, however the programmer(s) are working with them so that the pipeline works with the team’s abilities.

### Visual Studio

Risk: Low

Visual Studio is being used instead of Unity's built-in MonoDevelop. It is a standard for C-style programming languages and is a very useful debugging platform which Unity has full integration with. The programmer(s) are more familiar and at ease using Visual Studio than MonoDevelop.

### Git

Risk: Medium

Git is a version control software which allows the team to host project files on a remote site (the cloud). It will help the team to organize files in a way which all team members can access. While the team is being introduced to Git, at least one programmer has extensive experience in the software and is confident he can support the team as their guide.

All project and document files will be stored long term on Champlain's Pineapple site (built on Redmine). In order to do so, team members will use SourceTree or GitKraken, which are a windows/mas. These clients’ purpose is to aid users in the pulling/pushing of content to/from a remote server without use of the command line. In our case, it will be utilized so that team members can pull/push documents and project content between their local machines and Champlain’s Pineapple server. This will make using Git easier as it allows team members who are not as familiar with the inner workings of Git to use a graphical interface instead of command line, which can be confusing when first introduced to Git.

#### File Structure

General Naming Schemes:

* Docs
  + [DOCUMENT] \_Milestone[X].[ext]
  + i.e. TechnicalDocument \_Milestone1A.pdf

File Structure

* docs – holds any documents for safekeeping in the cloud
  + art - artist docs
  + designing – designer docs
  + production – producer docs
  + programming – programmer docs
* project – holds all project content (this is the Unity directory)

#### Repository Information

##### Master (Root/Trunk)

The master branch – under most circumstances – should not be committed to. This is the main source of information for the game. Builds must be built from this branch.

**Exceptions**: You can commit directly to the master under these exceptions

* Fixing a bug < 1 line of code (see Branches)
* Tweaking 1-2 variables in the unity inspector
* Documents

##### Branches

**Types**:

* Features
* Bugs

**Naming Scheme**: [TYPE]/[FEATURE\_NAME]

**Examples**:

* feature/unit\_placement
* hotfix/units\_not\_killed
* release/v1.0.0

**When to use**:

All features must start in feature branches. All feature branches should have an associated feature issue in pineapple of the same name.

Large bugs (i.e. >1 line of code) must also be put into feature branches, but do not need an associated feature issue. Rather, these should be individual tasks under one Bug feature issue.

Bugs which do not meet this description can be directly pushed to the master branch.

**Exceptions**:

When there is only one person active in the repository (i.e. during setup) branches can be ignored. This should only persist for 1 week at maximum.

**How to Use**:

1. Create feature branch
2. Make changes
3. Commit to branch
4. Test locally
5. Check for merge issues with master branch
   1. Fix merge issues
6. Merge with master
7. Delete feature branch
8. Tag master (see Tags)

##### Tags

**What are tags**: Tags are a way to view changes in the repository without digging through the changelog

**When to use**: When a feature branch is merged with master, when a bug is fixed, when a release is built.

**Naming Scheme:**

* *Bugs*: X.X.\_
  + Find last X.X.X tag, increment 3rd digit
  + Examples:
    - 5.1.8 -> 5.1.9
    - 0.0.9 -> 0.0.10
    - 41.542.2152 -> 41.542.2153
* *Features*: X.\_.X
  + Find last X.X.X tag, increment the 2nd digit by 1, set 3rd digit to 0
  + Examples:
    - 2.4.1 -> 2.5.0,
    - 1.9.0 -> 1.10.0
    - 8.25.8432 -> 8.26.0
* *Builds*: \_.X.X
  + Find last X.X.X tag, increment 1st digit, set 2nd and 3rd digit to 0
  + Only done by programmers or designers
  + Examples:
    - 7.1.0 -> 8.0.0
    - 0.63.52 -> 1.0.0
    - 12.5432.534263 -> 13.0.0
* *Milestones*: Milestone\_
  + Find last MilestoneX tag, and increment the digit X
  + Should ONLY be done by the producer
  + Examples:
    - Milestone1 -> Milestone2
* *QA*: Milestone\_-QA\_
  + Find last QAX tag, and increment the digit X
  + Keep the milestone digit from the last milestone tag
  + Examples:
    - QA1 -> QA2

### Photoshop

Risk: Low

Both the artist and designer have experience with Photoshop. Photoshop is one of the artist's preferred methods of 2D asset creation.

### Redmine/Pineapple

Risk: Low

Champlain’s Pineapple site is run on Redmine, and hosts all Milestone documents, the Git repository, and various information about the project in the wiki.

### Skype

Risk: Low

Skype is a text, voice, and video chat software which all team members are familiar with. This will be used to include members of the team who cannot be physically present at a team meeting.

### Discord

Risk: Low

Discord is a simplistic and easy to access text chat client which all team members will be using to communicate in an efficient manner. It is hosted by the company, and therefore does not need to be self-hosted.

### Maya and 3DSMax

Risk: Low

Artists know one or both platforms very well. These tools are used to create 3D models and art, and are self-sufficient programs. Artists can easily use these tools to export 3D models for usage in Unity.

## Delivery Platform

Risk: Low

We are building the game for the Windows PC platform. Unity natively exports to the Windows platform, and it is the easy to manage and maintain. We will be natively including support for Xbox controllers and keyboard/mouse controls. Since the game is multiplayer, we will be adding support for both dedicated servers as well as peer-to-peer play. Dedicated servers will allow for a separate computer to handle communication between players, whereas peer-to-peer structures will allow for players to easily spin up new games in which they can play together.

## Game Mechanics and Systems

See risk analysis of game mechanics and systems under separate documents.

## Art Pipeline

### Concept to Digital

1. Discuss / Brainstorm with group
2. Artist and Programmer discuss implementation
3. Artist sketches either a physical or digital illustration
4. Artist gets the go ahead from Designer (if not, go back a step)
5. Asset made in either Photoshop or Illustrator

### Digital to Repository

1. Asset is saved
   1. PNG Format
   2. Resolution of 72 pixels per inch and flexible pixels per unit
   3. Files will be organized by a folder structure and named appropriately
      1. Formatting: “Assets/Resources/Sprites/*type*/*asset*.png”
      2. i.e. “Assets/Resources/Sprites/Background/ground.png”
2. Asset is copied to Repository folder under "./project/Assets/Resources/", and to the appropriate subfolder
3. Asset is stashed, committed, and pushed to remote repository to any appropriate branch that is not the master

### Repository to Unity

1. Unity will auto import any assets at the specified path when opened
2. Artist or Designer opens sprite/asset in Unity inspector
   1. Modifications made to pixels per unit or slicing made as needed

### Unity to Game

(Mostly handled in the Design pipeline)

1. Drag into world space OR make prefab
2. Pixels per unit, various materials, and sprite slicing are all handled via the unity inspector

## Design Pipeline

### Unity Engine

* **Assets.** All assets can be found under Unity’s asset inspector
  + **Materials**: materials
  + **Prefabs**: all Unity prefab objects
  + **Resources**: all art assets
  + **Scenes**: Unity scenes
  + **Scripts**: custom scripts
* **Prefabs.** Prefabs should be made when GameObjects will be reused - specifically for units and spaces.
  + These can be made by dragging an object into the prefabs folder in the asset inspector
  + A brief list of example prefabs:
    - Level – contains most level-to-level singleton scripts like the EventSystem and Mouse input controller
    - Overlay Controller – controls Wii input and crosshair-scene interactions. Does not get destroyed when new scenes load, as the Wii object must stay intact.
    - Pawn – The player prefab, which contains the camera and parallaxed backgrounds.
    - Roads – various road prefabs which pertain to specific road curves. Contains colliders.
    - Enemies/Obstacles – all obstacle that the player would encounter are prefabs for easy usage.
* **Sprites.** Sprites can be dragged in from the asset inspector to the world window to be created as a GameObject
* **Colliders.** All interactable/collidable prefabs need a collider to interact with one another (and interact with player selection). Either consult the Unity manuals or a programmer.
* **Scripts.** Programmers will write custom scripts as necessary. These can be attached to their GameObjects by a) dragging the script on top of the GameObject or prefab or b) Adding a script component inside the GameObject Unity inspector (see Unity manuals).
  + **Public Variables**. Public variables will be utilized as much as possible to give other team members the ability to tweak values for gameplay and balancing. Examples of this include movement distance, attack range, team color, and if a unit will overtake other units on attack. Values can be changed in the Unity inspector when a GameObject with the script is selected. To add more variables, contact a programmer to discuss how it can be done.
  + **Guide.** A guide kept up to date with each commit can be found on the wiki under Team Resources. (A duplicate is below)

## Milestone Updates

### Milestone 1A

### Milestone 4

* Cody Douglas
  + [4/19] Fixed task board issues
  + [4/20] Updated the wiki
* Michael Andrews
  + [4/21] Level 2 road sign
  + [4/21] Ground enemy, flying enemy, and placeholders for environment assets
  + [4/24] Finished the character sprite, recolored enemy sprites, almost finished with a third enemy, and did the blood splattered jeep
  + [4/24] Pushed enemy sprites, character/jeep sprites (blood and normal), and 2 road sign sprites (a small one and a big one)
* Jennifer Carlin
  + [4/22] Mothership and speedometer UI
  + [4/22] Completed and implemented sprite for speedometer, searched for logo fonts, mocked up and completed 90% of assets for start screen, end screen, scoreboard, score entry, and pause menu
  + [4/22] Made a doc with mockups of every UI screen and put it on the wiki
  + [4/24] Pushed six building sprites, got two buildings in three states of crumbling each
* Conor Tully
  + [4/23] Merged Level Design into Develop
  + [4/23] Finished QA preparations
  + [4/23] Changed Menu start button scene pathing
  + [4/23] Merged UI to Develop
  + [4/24] Figured out values for mothership
  + [4/25] Updated GDD and did QA analysis
* Jake Ettinger
  + [4/20] poke the UI
  + [4/21] Added UI setup and Audio samples, including music
  + [4/23] Pushed UI
  + [4/25] Added audio and tied them to events for shooting, enemy death, and music.
  + [4/25] Pushed a version of the level with improved audio balance
* Mico Demo
  + [4/21] Starting on enemy prefabs
  + [4/21] Finished enemy prefabbing
  + [4/21] Level design
  + [4/22] Enemy placement
  + [4/24] Finished buildings, particle effect placement, made and implemented a particle for destruction of rubble
  + [4/25] updated VDD and GFD sections
* Dustin Yost
  + [4/21] Review the enemy movement
  + [4/21] Repo Management: Merged all art assets, particles, and enemy-AI
  + [4/21] HUD, Game manager, and Audio scripts
  + [4/23] Finished kill bumper tweaks
  + [4/24] Finished reviewing the mothership
  + [4/24] Polished mothership, Fixed input issues with mouse, Work on sand trap effect
  + [4/24] Added the winstate transition
  + [4/24]
    - Add score carrying over into death scene
    - Fix events for audio and score
    - Fix camera render distance
    - Merge audio and fix scene conflicts
    - Review mothership for jerky movement
    - Transition level scene into death scene
    - Implement Death scene
  + [4/25] Merged buildings and tweak build setup things
  + [4/25] Tweak build and perform quality checks before QA
* Cory Smith
  + [4/20] Got a little bit of stuff done on the enemy movement
  + [4/21] Finished script for node based movement
  + [4/22] Debris placement
  + [4/22] Worked on the mothership logic for the winstate
  + [4/23] Pushed the mothership

### Milestone 5

* Cody Douglas
* Michael Andrews
  + 5/2
    - Pushed debris asset, am working on moose now
    - Pushed the full moose asset, moose head, moose body, and tweaked the player sprite
* Jennifer Carlin
* Conor Tully
* Jake Ettinger
  + 5/2
    - Pushed finished pause menu
* Mico Demo
  + 4/26
    - Finished particle for flying enemy death, they blow into chunks and wherever the chunks hit a stain is made
    - Fixed fires in level
    - Added smoke and ash particles to menu
* Dustin Yost
  + 4/26
    - Update tech doc
      * Logs
      * Sprint 3 feedback
  + 4/28
    - Added background audio pitch changes (script)
    - Added background fire alpha animation (controlled entirely by Animation file)
    - Added the meter for measuring distance to shadow
* Cory Smith
  + 4/29
    - Shooting enemies & divebombers are pushed to the updates enemies remote, just finished them