

# Everybody Fits In

- *Technical Risk Assessment* -



Andrew Rimpici and Alexander Hubble

- EGD-220-04 -

## TABLE OF CONTENTS

<b>Difficulty Scaling System</b>	<b>3</b>
Document Scaling System	3
<b>The Delivery Platform</b>	<b>4</b>
Main Delivery Platform	4
Other Delivery Platforms	4
<b>The Development Environment</b>	<b>5</b>
Programming Environment with Unity	5
Resource Environment	6
Version Control	7
<b>Game Mechanics and Systems</b>	<b>8</b>
Game Systems	8
Game Mechanics	9
<b>The Art Pipeline</b>	<b>12</b>
Creating the Art	12
Using the Repository	12
Loading the Art in Unity	15
<b>The Design Pipeline</b>	<b>16</b>
Unity Engine	16
Editable Components	17
<b>Milestone Updates</b>	<b>18</b>
Milestone #1	18
Milestone #2	19
Milestone #3	20
Milestone #4	21

# DIFFICULTY SCALING SYSTEM






## Document Scaling System

### Scaling Overview

- This document uses a 1 through 5 difficulty scaling system where 1 represents very easy, and 5 represents extremely difficult.

### Difficulty Representation

- The difficulty will be represented using puzzle pieces shown below:

	<b>Represents 1 out of 5</b> Very Easy - No trouble to implement
	<b>Represents 2 out of 5</b> Easy - No trouble to implement but may take more time than usual
	<b>Represents 3 out of 5</b> Medium - More thought than usual went into planning and implementing
	<b>Represents 4 out of 5</b> Moderately Difficult - A lot of planning and time went into implementing
	<b>Represents 5 out of 5</b> Very Hard - Very difficult to plan and to implement. Most likely would take too long to implement before a deadline

# THE DELIVERY PLATFORM

## Main Delivery Platforms

**Computer - Windows, Mac, Linux, and Tablet** -



- *Everybody Fits In* is primarily designed to be developed and played on computers and tablets. This is mostly because *Everybody Fits In* is a game targeted for schools to use in classrooms with kids. American classrooms can range from owning windows laptops, to iPads and we want to be able to accommodate all of these cases. The easy drag-and-drop interface of the game makes it easy for the programmers to adapt the game for either tablet or computer based systems.

## Other Possible Delivery Platforms

**Mobile - iPhone, Android** -



- Another potential delivery platform for *Everybody Fits In* is mobile handheld devices. Since *Everybody Fits In* is a drag-and-drop based game, the mechanics can easily be converted to a touch based system allowing for mobile devices to play the game. IOS and Android devices are the most popular devices in the mobile market right now, so these systems would be beneficial to target.

# THE DEVELOPMENT ENVIRONMENT

## Programming Environment with Unity

*Using The Unity Engine* - 

- The game will be developed in the Unity Game Engine. Unity comes with a lot of built-in features that are right at the developer's disposal, which eliminates the



need to construct a game engine specific to the game from the ground up, thus *saving a great deal of time*. A majority of the team is comfortable using the Unity Engine which translates to a more efficient workflow.

# THE DEVELOPMENT ENVIRONMENT

(Continued)

## Resource Environment

*Adobe Photoshop* - 

- All of the art assets used for the game will be created in the Adobe Photoshop art program. Photoshop is a very widely known and well supported graphics tool



that allows for users to create things of all shapes and sizes. It can be used to draw or edit photos. The program mainly works in a raster format and not vector format which works perfectly for *Everybody Fits In* as the assets will be pixel based and not vector based.

# THE DEVELOPMENT ENVIRONMENT

## (Continued)

### Version Control

*Subversion* - 

- The choice of version control for this project is Subversion (SVN). In Pineapple, there is a subversion repository linked where all of the team members can monitor and



update the game files. The use of Subversion will help the team make sure all of the game files are up-to-date and where they should be in regard to meeting deadlines.

- In the repository on pineapple there is a root folder called “Unity Tree” that holds all of the different branches and builds for the game. This makes it more convenient to pinpoint certain builds that might have bugs in them in addition to keeping backups and snapshots of different build milestones.

# GAME MECHANICS AND SYSTEMS

## Game Systems

### Win detection -

- The win condition in *Everybody Fits In* is to complete a tangram style puzzle, so a win is when the pieces fit into the puzzle. To detect this state, the game controller object holds two integer variables. The number of total pieces and the number of required pieces. When the gameobject is created, the script attached will call a function on the game manager's script *addNumOfPieces()* which adds to the total number of pieces in the puzzle.
- Next whenever a piece detects that it is in a correct spot, it will run another function in the game manager's script *addNumOfCorrectPieces()* which adds to the number of correct pieces.
- Finally, the game manager runs the *checkWin()* function within the update function. This function checks to see if the two variables mentioned above are equal to each other, when this is true it is indicated that the puzzle has been completed, therefore the winstate has been achieved. With this check, the program can safely check for a win condition as well as reuse the same script for multiple puzzles.



# GAME MECHANICS AND SYSTEMS

## (Continued)

### Game Mechanics

*Drag and Drop (control scheme)* - 

- Drag and drop is the control scheme we will be using in *Everyone Fits In*. This is preferable for several reasons:
  - It's easy for elementary level children to use.
  - It can be implemented easily on devices such as computer and tablets.
  - It's easy to implement within the game itself.
- All pieces are clickable with a bool *isMouseDown* inside stating if the object is clickable or not. The unity functions *OnMouseDown* and *OnMouseUp* toggle *isMouseDown* with their respective states. Within the object's update function, there is a function called *moveObject()*. This function will automatically move the game object to the mouse coords if the *isMouseDown* bool is true, or nothing if false. This allows the user to move any piece with the mouse.

# GAME MECHANICS AND SYSTEMS

## (Continued)

### *Snap to spot* -

- This mechanic serves two primary functions within the puzzle, to help check the win state and to help younger players fit pieces in accurately. As of the current prototype it works simply. Once the held piece is dropped and is within a snap spot AOE, it will snap to the center of that AOE.
- The programming in this is shared between two scripts, the primary script for the pieces and AOE spots. There are two key parts to this function within the piece script, the check function *checkForSnap()* and the on trigger functions. The on-trigger functions work simply; if there is an entered collision of an AOE piece (detected through the tag on the object) that object will be added to a list of AOE's the piece is currently in, when the trigger is exited, the AOE exited will be removed from this list. Within the check function, there is a series of if statements within a loop which goes through each object within the list of AOE's the piece is currently inside of. Then for each AOE it will check if it's not null, the piece is enabled (if it is eligible to be used again), and if the mouse is currently holding said piece. Then it will make sure that the object within the list is a AOE (using the unity

tag system). Then finally the check calls a script within the AOE's script, this will check to see if the object that entered the AOE is eligible to fit within the AOE, this is done by checking the tag of the piece against a string within the AOE's script. Then if that script returns true, the piece will call the addNumOfCorrectPieces() within the game manager, snap the piece to the center of the AOE's collider, and disable the piece.

### *Rotation* -

- With any complicated piece (L piece, Z piece, ect.) the player would be able to rotate by hovering over and either pressing the spacebar, right clicking, or tapping the screen with a second finger if on mobile.

### *Pre Snapped/Anchor Pieces* -

- To help the understanding of the purpose of the game, some of the "Special Needs" pieces would be automatically snapped into the board. To do this, there would be a global bool variable that if true, would automatically call the addNumOfCorrectPiece() within the gameManager script to indicate that it is correct, this would also disable the piece to prevent the piece from moving.


# THE ART PIPELINE

## Creating the Art

- The pieces in *Everybody Fits In* are scaled on a 100x100 scale, with the base scale being a 100x100. What this means is that for any piece the scale can be figured out by the shape of the piece. For instance if we need a rectangle, the dimensions would simply be found by adding 100 in whatever direction the rectangle is facing, so in this example it would be a 100x200 piece. Any non-square objects will always fit within a 100x100 canvas. For any odd pieces (such as the L and Z pieces) the canvas will be an even scale, however any unused space should be transparent.

## Using the Repository

### Step 1 - Download TortoiseSVN

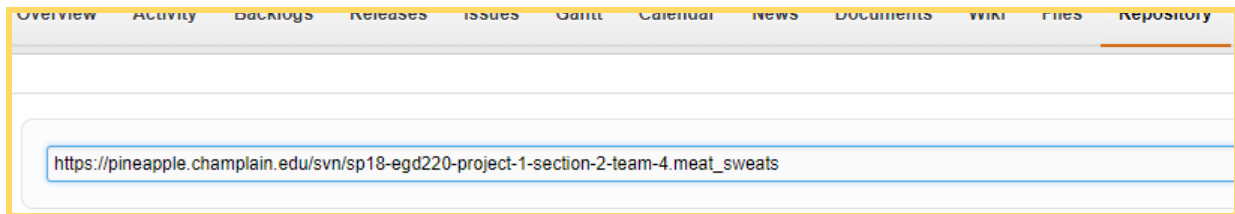
- The first step is to download TortoiseSVN. TortoiseSVN is  a client software that allows the user to upload documents and files to a server for other members with access to view and manipulate. TortoiseSVN is free, easy to learn, and easy to use.

# THE ART PIPELINE

## (Continued)

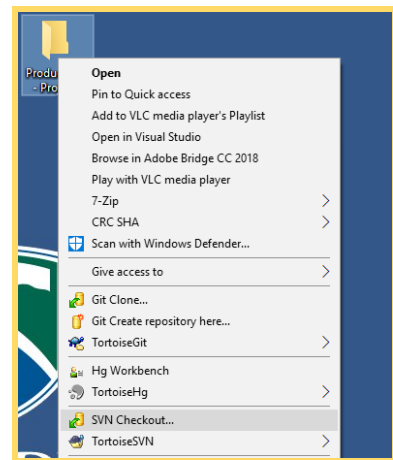
### Step 2 - Navigate to Pineapple

- Once downloaded, the user should log into Pineapple and under the team's "Repository" tab, there will be a URL.



### Step 3 - Create TortoiseSVN workspace folder

- The user should copy the URL to the computer's clipboard by using the hotkey CTRL + C. Once that is done, the user should create a folder on their desktop or other location of choice. Right-click on the folder and choose the option called "SVN Checkout."

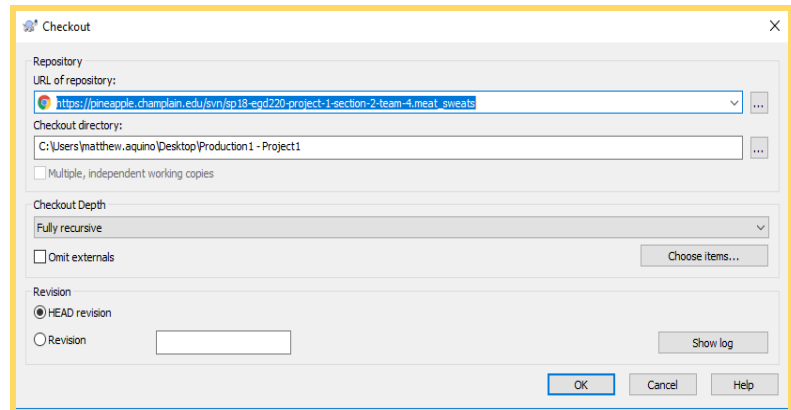


# THE ART PIPELINE

## (Continued)

### Step 4 - Paste Pineapple URL in TortoiseSVN

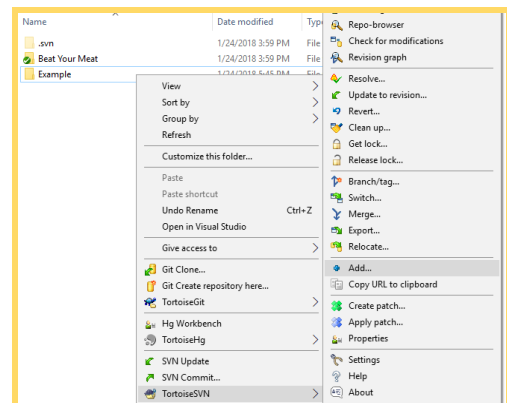
- The URL from the previous step should automatically be placed in the window that pops up. The user should click “OK” and log in with



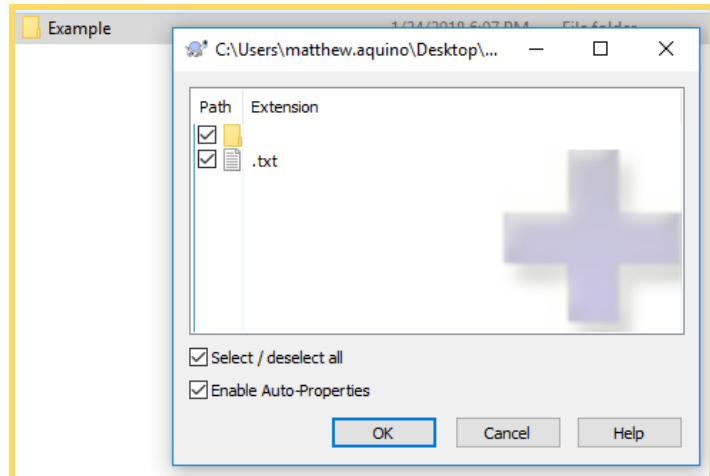
their Pineapple credentials when prompted. This will load all folders in the repository into the folder they’ve created.

### Step 5 - Add Files/Folders to the Repository

- If the user wishes to add a new folder to the repository, all they simply have to do is copy the new folder into the repository folder, right-click on it, highlight the “TortoiseSVN” option, and select “Add.”



- Select the folders you wish to add, or choose specific files to add, and click “Ok.”



### Step 6 - Commit New Files to the Repository

- Finally, simply commit the folder by right-clicking on it, and selecting “SVN Commit...”. Click “Ok” and the folders should be uploaded to Pineapple. It is very important to always update and commit any new versions of work so that the team is always up-to-date.

### Loading the Art in Unity

- The art will be loaded directly in by the designers and / or programmers once the art has been uploaded into the repository. For any odd shapes, the collider will have to be manually set using the polygon collider.

# THE DESIGN PIPELINE

## Unity Engine

### Implementing Puzzles -

- Within the program, there is a specific snap AOE (Area of Effect) for each piece, these pieces will need to be placed in any spot within the puzzle that the specific piece can fit into. For instance, if there is a spot where the L piece can be fitted into that AOE will go there. However since 1x1 squares can also fit within the spots on the L piece the AOE's will have to be put where appropriate. With the way the game is programmed, it is safe to have multiple AOE's overlapping each other. Any pre-existing pieces or AOE's are saved as prefabs for easy use. If any piece should be unmovable, create an anchor game object to represent this piece and have no AOE's within the anchor's space.

### Rotating in Engine -

- Each piece that can be rotated has a public enum (under the Shape Rotation Script) within the bound piece of the game object, this enum will showcase what direction the piece is currently facing. The AOE's also have this enum, however its located on the object itself.



# Design pipeline

(Continued)

## Adding New Pieces -

- Each piece consists of two parts, a bound and the art. The art part of the piece handles the rendering and any effects on the individual piece. The bounds part holds any scripts necessary and the colliders. AOE's just hold the polygon collider and any necessary scripts. The only exception to this is abnormal pieces (such as L and Z pieces) that require multiple rectangle / square pieces to fill in, however the collider will be edited to match the sprite.

## Editable Components

Prefabs for Designers		
Pieces	AOEs	Anchors
<b>Editables</b> <ul style="list-style-type: none"><li>- Transform</li><li>- Sprite Renderer</li><li>- Piece Script</li><li>- Shape Rotation Script</li><li>- Shape Scale Script</li><li>- Polygon Collider 2D</li><li>- Rigidbody 2D</li></ul>	<b>Editables</b> <ul style="list-style-type: none"><li>- Transform</li><li>- Sprite Renderer</li><li>- Polygon Collider 2D</li><li>- Snap Spot Script<ul style="list-style-type: none"><li>- <i>Required objects should be set to the tag of whatever object fits inside</i></li></ul></li><li>- Shape Rotation Script</li><li>- Shape Scale Script</li></ul>	<b>Editables</b> <ul style="list-style-type: none"><li>- Transform</li><li>- Sprite Renderer</li><li>- Collider 2D</li><li>- Rigidbody 2D</li></ul>

# Milestone Updates

## Milestone #1

### Deliverables:

#### *Artist Concepts*

- Three styles for possible art direction were created.

#### *Visual Design Document (VDD)*

- Guide for core mechanics.

#### *Game Rules Document*

- Rules pertaining to the game were outlined.

#### *Physical prototype*

- Board was created and tested.

### Goals for Next Milestone:

- Create a functioning digital prototype that effectively conveys the game's core mechanics

# Milestone Updates

(Continued)

## Milestone #2

### Deliverables:

#### *Digital prototype*

- The digital prototype is functioning, however there is no way to make multiple levels.

#### *Art Style*

- Solidified the art style for the game

#### *Technical Plan*

- The technical plan has been started and is currently being iterated upon.

### Goals for Next Milestone:

- Simple cutscenes
- Code in multi-level functionality
- Addition of more levels
- Functionality to unplace puzzle pieces that have already been placed on the board

# Milestone Updates

(Continued)

## Milestone #3

### Deliverables:

*TBA*

### Goals for Next Milestone:

*TBA*

# Milestone Updates

(Continued)

## Milestone #4

### Deliverables:

*TBA*