



Technical Document and Risk Assessment

Authors: Ethan Heil, Chris Foster

Team Members: Leah Gimbutas, Leah Peterson, Rhys Frampton, Mac Soulsby

EGD220-06

Sprint 3

Table of Contents

Table of Contents	2
Development Environment.....	3
Unity	3
Git.....	3
Overview of Branching	3
Basic Git Workflow	4
Delivery Platform	4
Main Delivery Platform	4
Other Delivery Platforms.....	4
Logical Flow Diagram.....	5
Game Mechanics and Systems	6
Grid Spawning	6
Player Initialization	6
Character Selection/Color Assignment.....	6
Generate Player Inventory	6
Gold Generation.....	7
Gold Movement	7
Turn Playing Systems	7
Beaver	8
Win/Loss Systems.....	8
Art Pipeline	8
Design Pipeline	9
Milestones	9
Sprint 1	9
Sprint 2.....	9
Sprint 3.....	9

Development Environment

Unity

O' Dam will be developed in the Unity Engine. Specifically, we are using Unity 2019.2.8f1. There is a myriad of reasons for why Unity is a good choice to create *O' Dam*. To start, Unity has a strong foundation of 2D development tools. Since *O' Dam* is a 2D game, Unity is a solid choice for developing it. Second, Unity is free to use, well documented, and has a large community of people using it for both personal and professional projects. This makes it very easy to find information about a problem if a programmer runs into trouble. Third, Unity's built in Visual Studio and Git integration allows for seamless and easy workflow. Fourth, Unity allows for us to develop and build our projects for other platforms. This is useful later down the road if we want to develop *O' Dam* for Mac, Linux/Unix, IOS, or Android. Lastly, all of our team members have had some experience working with Unity in the past. Since our team is familiar with the engine and how it works, it is much easier for us to make changes to our game.

Difficulty: Low

Git

Our team will be using Git for version control with our repository hosted on RedMine. Since several members of our team are familiar with Git, we have been able to implement branching as we add in new features. This allows for a release build of the game to be available while we work on the development build.

Difficulty: Moderate-High

Overview of Branching

- Master Branch
 - Branch meant for stable builds and the finished/polished product.
- Dev Branch
 - This is the work in progress branch where all the unstable code and unfinished assets go.
 - Code for specific features should not be altered in this branch.
- Feature Branches
 - Feature Branches are created off the dev branch to work on a specific feature.
 - These branches are mostly for the programmers or designers to work on specific game features (UI, Character Selection, Movement, etc.).
 - The programmers/designers should push their work to the feature branch that they are working on until that feature is complete.
 - Once a feature is complete it should be merged into the dev branch.

Basic Git Workflow

1. `git status` (Checks for any changes on your local machine)
2. `git fetch` (Checks for any changes in remote repository)
3. `git pull` (If fetch shows changes, pull the changes to your local machine)
4. `git status` (Checks for any changes on your local machine)
5. `git add <file/folder name>` (Stages files for commit)
6. `git commit -m 'message'` (Commits files to be pushed, add a commit message so you know what was worked on)

Example commit messages:

“Adding README.txt”

“Adding move function to player.cs”

“Adding playerSprite.png to Assets folder”

7. `git push` (Pushes files to the remote repository ON YOUR CURRENT BRANCH)

Further git workflow information can be found in the repository's README.txt. Workflow information contains different steps for the different disciplines (aka. Artist, Programmer, Designer)

Delivery Platform

Main Delivery Platform

O' Dam is primarily designed to be played on the PC on the Windows operating system. Thanks to Unity allowing for multi-platform development, making a build for Mac and Linux/Unix will not be a huge hassle. Our game's mechanics focus heavily on a point-and-click system. Both players take turns selecting and placing their dam pieces in order to get the gold to their side of the river. Although *O' Dam* was originally designed for PC, it can be easily scaled to other platforms.

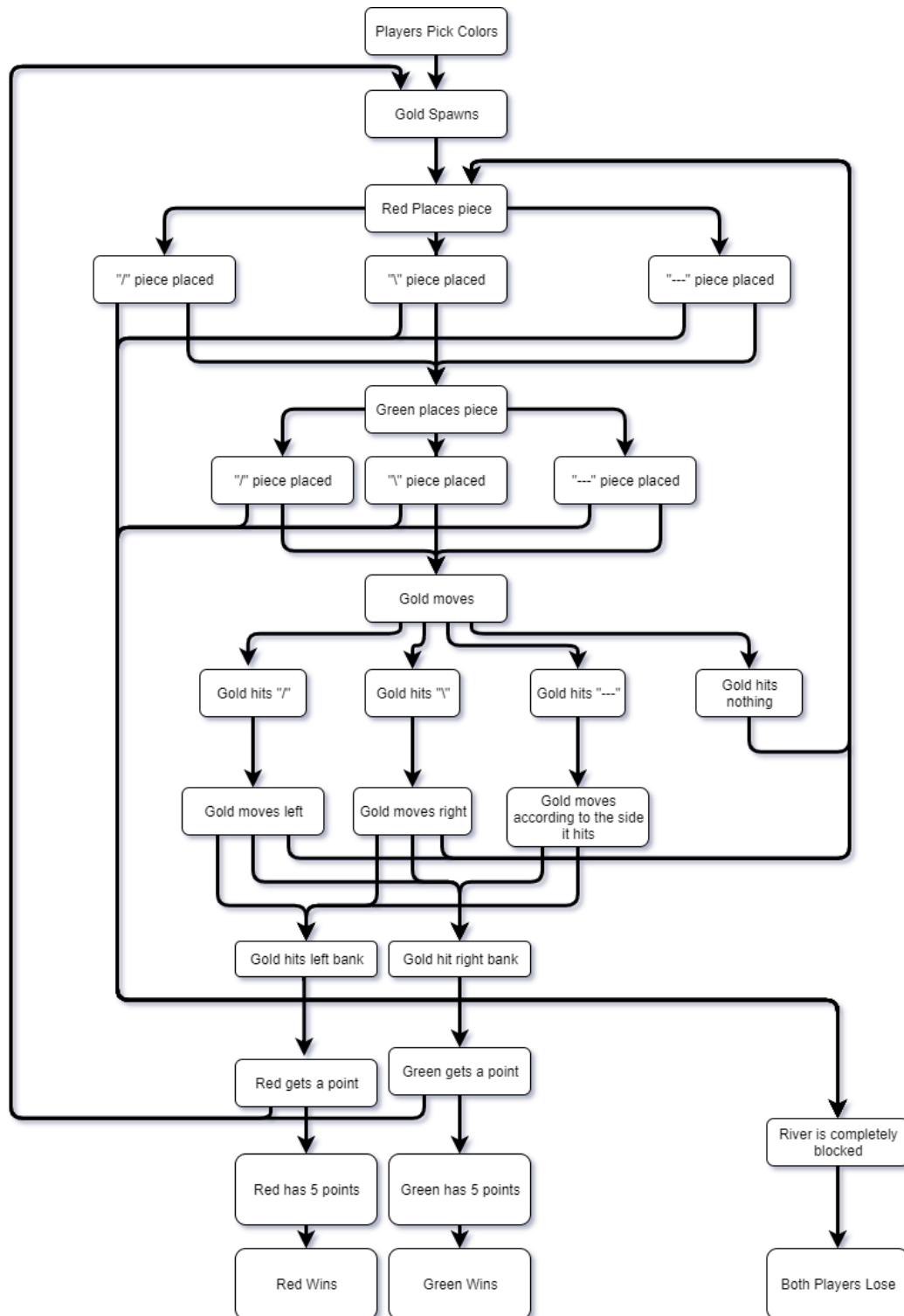
Difficulty for Main Delivery Platform: Easy

Other Delivery Platforms

Other potential delivery platforms for *O' Dam* are mobile and touch screen devices. Translating our point-and-click system to a point-and-touch system would be very simple. IOS and Android are currently the most popular devices in the mobile market right now, so developing these systems in addition to the PC's systems could be good if we decide to deliver to mobile platforms.

Difficulty for Other Delivery Platforms: Easy-Moderate

Logical Flow Diagram

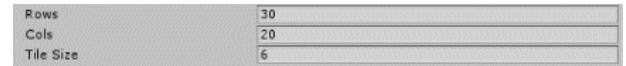


Game Mechanics and Systems

Grid Spawning

Grid

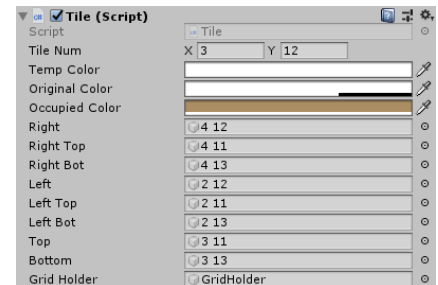
O' Dam will be played on a grid system which will be instantiated at the beginning of every game in an X by Y fashion. There will be an empty game object in the scene with the script Game Manager which will hold all the code pertaining to the grid. Designers will be able to edit the grid in the Unity editor with the Rows and Cols variables. The tiles that will spawn will be a prefab in Unity that can also be edited by designers if need be. The grid will have the sprites for diagonal and flat dams, as well as the original tile sprite, as variables in order to change the tiles sprites when placing and destroying dams.



Rows	30
Cols	20
Tile Size	6

Tiles

Each tile will have a number associated with it representing its location in the grid. In addition, it will have references to all the surrounding tiles to assist with grid checking, as well as a reference to the Grid Manager. This allows for tiles to react to mouse input according to what state their neighbors are in (dammed or not). In addition, they will contain true or false variables to help determine what state they are in. This will assist in gold movement as it can be determined if they are dammed, and if they are, what part of the dam they are in. The color variables help to show when a player is hovering their mouse over a tile and will show the type of dam being placed.



Player Initialization

We will be using Unity's scriptable objects to keep track of both player's data throughout the game. Scriptable objects are a nice way to access information without having to have an object in the scene. Since our players do not actually control a character while playing, storing data this way will prevent our scene from being cluttered with unnecessary objects. Each player has a name, assigned color, current state (either idle or active), and another scriptable object that stores the player's inventory. All of these will be set to default values at the start of the game.

Character Selection/Color Assignment

At the start of the game each player will choose one of two characters, either a farmer/cowboy or a rich man/industrialist. Once a character is picked it will not be available for the other player to choose. Both player's colors will be set in their respective scriptable objects (green for player 1, red for player 2).

Difficulty: Low

Generate Player Inventory

The inventory system for *O' Dam* is very simple. A player's inventory will consist of a scriptable object holding three integers, meant to keep track of the number of straight, diagonal, and

dynamite pieces respectively. Once the counter for a specific piece reaches zero, the player will not be able to use that type of piece for the rest of the game.

Difficulty: Low

Gold Generation

At the beginning of each round, gold is spawned randomly at the top of the river. The Grid Manager will contain references to each spawning tile. Because the grid is generated at runtime, the locations will be found according to grid dimensions such as the width. The tile it starts on will always be at the top row of the grid and will spawn at a random x location.

Difficulty: Low

Gold Movement

As the players place their dams the gold moves down the river, reacting to the various obstacles in place. If unimpeded, the gold will move three spaces down. However, if it does, it will react accordingly. Upon hitting a diagonal dam, the gold will move in the opposite direction with the remaining number of moves it has, until stopping, hitting another dam, or reaching a player's riverbank. Upon hitting a flat dam, it will move according to the side of the dam it hits. If it hits the right side, it will move to the right. If it hits the left, it will move to the left.

Difficulty: Moderate

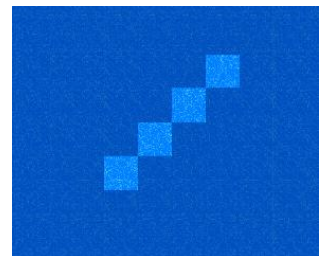
Gold Reset

If the gold reaches the bottom of the screen without making it to either riverbank, it will be reset randomly at one of the two starting locations.

Turn Playing Systems

Placing Dams

When players hover over tiles while placing dams, the tiles that would be affected appear highlighted. The tiles that are highlighted are chosen in relation to the tile that is being hovered over, using the neighboring tile references. This helps the player to understand what they are doing and allows them to plan out their placement for upcoming turns. The type of dam being placed will be chosen by the player with each turn and will be controlled by the Grid Manager.



Difficulty: Moderate

Dynamite

Dynamite will allow a player to destroy their dam or their opponents. This will work the same way placing a dam will, however, it will only highlight the tiles in the dam that is being hovered over. One dynamite stick will destroy one dam piece.

Difficulty: Low

Beaver

Every round will have a one and three chance to spawn a beaver. The beaver will spawn on the grid below the screen at a random x location. It will swim up the river, moving from tile to tile, destroying any dam that it crosses. It will use a reference the Tile script that it spawns on to navigate up the river.

Difficulty: Moderate

Win/Loss Systems

Gaining Points

A player will gain points when gold has reached their riverbank, or their side of the grid. The surrounding tiles will be used to determine if the tile has reached a side of the grid. In addition, the tiles number will be used as well to ensure that the gold has reached the side. Once a player has reached 5 points, they will win the game.

Flooding the River

At the end of each turn the river will be checked from left to right to see if it is flooded. It will be considered flooded if a continuous path of tiles can be made using diagonal tiles as well as tiles to the immediate sides.

Difficulty: Moderate-High

Art Pipeline

All the art assets for this game will be produced by the artist in Adobe Photoshop for a 16:9 (1920x1080px, Full HD) resolution ratio. Adobe provides valuable image editing tools and support for drawing tablets allowing artists to get the most out of it. Though it does come at a cost our artists all already have access to it as well as having access to the school labs.



All art assets will be created for a 16:9 aspect ratio and will be uploaded to the repository on the development branch. Assets will be placed directly into the Unity projects “Sprites” folder. This allows for unity to load them without any further work and makes implementation very easy when programming. After the art has been placed in the “Sprites” folder, the artist will follow the basic git workflow outlined earlier. The artist will only push their assets to the dev branch to avoid any merge conflicts.

Files will be exported as PNG files using the camel case naming convention (ex. nameName01, dynamiteSprite01, redProspector03). PNG files allow for transparent background which is important because it removes unnecessary white space. This could be done with a shader potentially but the work it would take to do it work be far greater and it would also lower performance.

Design Pipeline

For this project, mechanics are discussed amongst the group and are either accepted or rejected. After brainstorming, the new mechanics are thoroughly discussed with the programmers to be sure that they can be implemented effectively. Ideas that have been chosen to be included in the game will then be given to the programmers to implement in code. All scripts will be made so that designers can change the variables in the Unity editor without directly editing code. This allows the designers to test different numbers for balancing and game feel. After being implemented the new mechanics will be presented to testers to see if they work well within the game, or if any new mechanics pique the tester's interests. Once the mechanic is fully understood and communicated to the whole team, we then implement art assets for the new mechanic if they are needed.

Milestones

Sprint 1

- Repository was made and set up with a README.txt file
- Physical Prototype created and tested
- Unity project created
- Rough prototype of core mechanics
- Preliminary research on water effects and research

Sprint 2

- Digital prototype iteration
 - Tile/barrier placement coded (not bug tested).
 - Grid layout made with Grid Manager script (not tested).
 - Player color selection prototyped (not tested).
 - Rough game loop implemented
- Clarified git pipeline and added a GITRULES.txt file to the repository for reference
- Technical Document Drafted

Sprint 3

- Finalized digital prototype
 - Refined game loop
 - Implemented art assets
 - Added working UI with art assets
 - Added water effects
 - New beaver mechanic
 - Added Win/Loss screen
 - Added background music
 - Added Title screen
- Update Technical Document