Unity

# Introduction

## Brief

This area will be added to as new knowledge of each section in acquired. I will also try a pair each introduction section with the GitHub commits. I good starting note is the unity development kit was almost made for non-programmers since even some of the most advanced tutorials are very simple in terms of programing knowledge.

## Unity Hub

This is a tool used to develop the unity game. The most obvious change compared to other IDE is its focus on a scene and not any form of code. Most IDE have some drop and drag feature but it normally isn’t the primary way you implement things in the IDE and only works for some languages.  
In unity it’s in reverse the drop and drag UI if the primary feature and the scripts or code are the addition.

With this knowledge I will need to sift through the UI elements and turn them into code. The UI elements are inherently hard coded and not dynamic but if I can generate UI element with code, I can break this inbuilt cycle.

## Working with Unity

The Scene and code division adds a very interesting problem especially for a puzzle game that needs dynamic generation. That problem is I don’t need the scene at all, the only potential us is for something like backgrounds or UI.

The ability to create a dynamic puzzle requires the creation of the scenes assets that can be scaled. For future note a 2d game assets would instead be duplicated many times.

Another way to look at this is each scene asset is a separate view interface. They can instance once created and manipulated; this is how normal Unity development works but on a much smaller scale.  
Pushed to the extreme one asset could serve the entire programs needs, like let’s say like moving grass.

(Unity, 2019)

## MoSCoW

A good idea of what I need to priorities in each prototype.

### Must

1. Instanced grid assets
2. Grid changed with image files

### Should

1. Update grid with changed on image file
2. Saving image changed back to an image
3. Click image tiles to select or unselect them

## Could

1. Progress image vs real image
2. Saving the progress image as a separate image file
3. Loading/checking for save file image

## Would

1. Add some navigation
2. Hints to help discover the image

# Scene Assets and Code

With reference to another Nonogram game I bought I managed to come up with some core concepts.

(BZ, 2019)

## Brief

The idea is that each asset serves a view or interface, I will create all the assets I need and then manipulate them with code. The assets are the view, the code is a model and the controller in the scene itself with its main script.

Another key aspect is the use of a picture to fuel the generation of the view. This process will be reactive so during the creation of images though an in-game editor will instantly update the grid assets and therefore be re-rendered real time.

The in-game editor will have to have some logic to find out if a solution is possible and how many ways it can be solved which can then assign a difficulty, but this will be a much later addition

## Binding Assets and Code

This can be done in one of two ways, to some respects it is one-way databinding either the code generates the asset, or the script references an existing asset.

1. GameObjects can be created, components added and then instantiated.
2. Assets can be created; the script creates public variables and then the inspector can assign them to those Assets.

Some assets like Tilemaps have internal storage for each tile so once the public variable is created and assigned you can then set each tile individually on the Tilemap.

## Clickable

Buttons and other UI element have events built in already but in terms of game speed we want to use the smallest asset and rely on the most code.  
Code itself can listen to mouse inputs and retrieve the location of them, this can then be translated into events that can change lets say the color of a map tile.

# Dynamic generation

## Brief

The real goal is how can I make game assets that are not hard coded generate during runtime. Even things like projectiles will have to be generated during run time rather than always being placed.

Unity does this by creating GameObject then instantiate it into the game files, this then can be changed or copied based on your needs. This basic tutorial has a much simpler version of the Nonogram game asset I purchased.

(Games, n.d.)

## Grid and Tilemap

These are amazing GameObject’s that can first be set up with preferences inside the grid that will launch a script on start-up and then a Tilemaps can be laid out on top. These Tilemaps act as layers in photoshop.  
Both the Grid and Tilemap need to be created before the launch of the script and a public variable will hold a reference to them inside the script which will be assigned with the inspector(where the script is in unity)

## Dynamic generation location

The real question where the dynamic generation should go?

The example we have obtained places it inside the game main script at the start. This is okay but what if we want to reuse the code once, we have started the game for resetting screen without or changing the level.

If we think of the game script as the game model and the scene objects and view, then we can think of the dynamic generation as isolated State Hook that can be inherited by anything. This means a button could inherit the script to reset the picture when changing map or the grid can on start up.

The OOP approach should let the screen reset the puzzle from anywhere in the program. This should also be applied to any sort of Game-over or Win screen.