Documentation of CPS Game

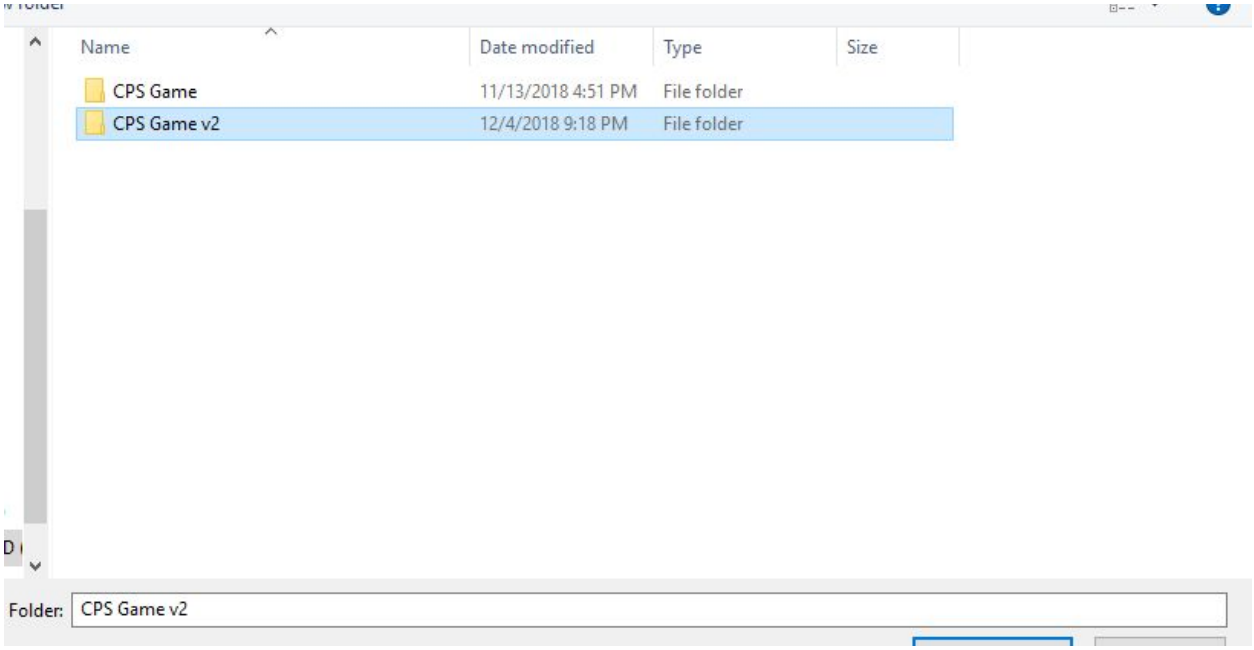
Unity

- How To Install -

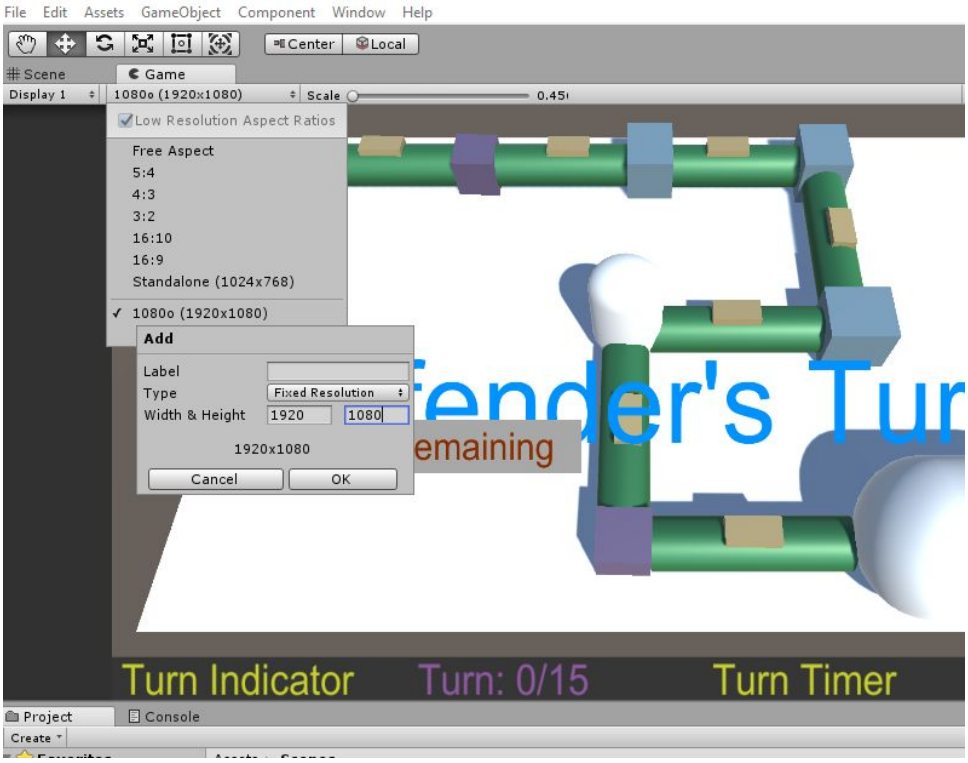
For this project, the program used was Unity3d and was coded in C#. Unity 3D can be downloaded here. This will download the unity hub were you can download the latest build of unity. We all used the Personal version of unity as it is free. When installing a new version of unity, make sure in the Platforms to install the WebGL Build Support. (We do not currently have anything that should be broken if new updates are released, but if the project appears to have problems, our latest working build we used was **Unity 2018.2.15f1** or **2018.2.14f1**).

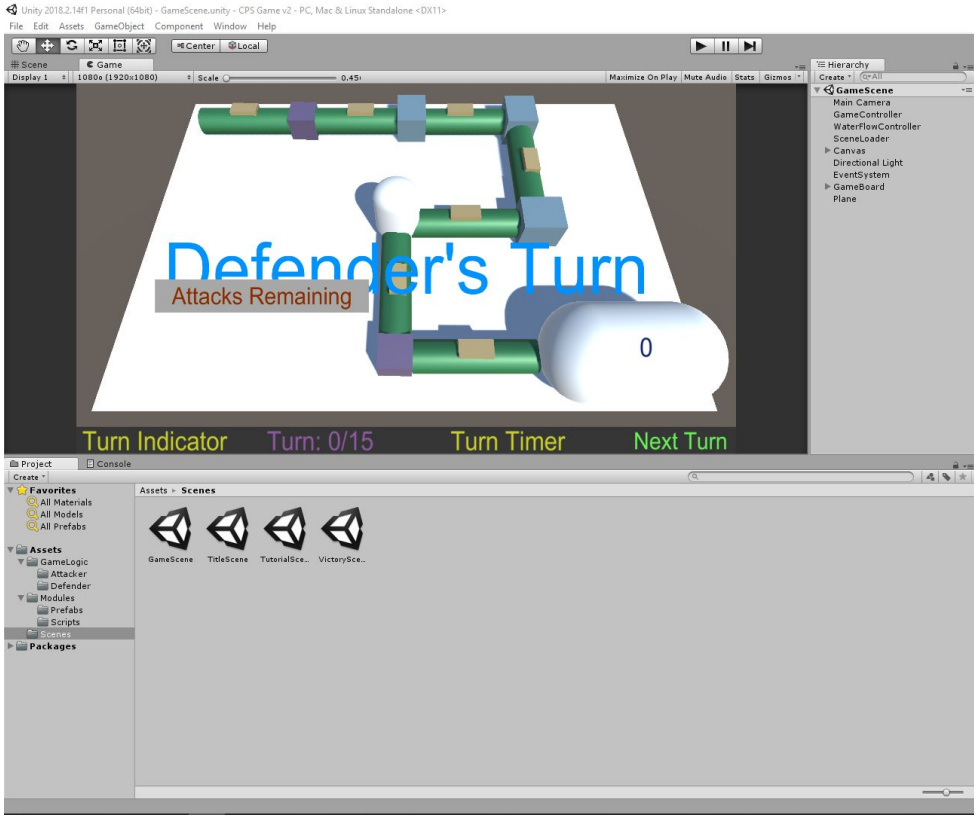
- How to Load Project -

Once you have installed unity and have the project code, using the Unity Hub, you can open the project. There will be an icon that says “Open” with an up arrow. Click on that and it will bring open the file explorer to select a project. The Project is the parent folder of where all the code is.



This file contains directly in it, all the code for our project and is what should be selected to open the project. Once the project is loaded up there will be a blank scene. To load up our game, in the Project tab under the display window, go to “Assets/Scenes” and drag “GameScene” to the Hierarchy tab. (Note: To ensure it works properly, make sure GameScene is the only scene loaded into the hierarchy. It will look like the image below). Another thing to make sure the game works properly is to set a set resolution for the game to run in. Our games resolution is expected to run at 1920x1080 and if it is not, the UI will not work properly. (See image below on how to set it).


File Edit Assets GameObject Component Window Help
Center
Local
Scene
Game
Display 1
1080o (1920x1080)
Scale
Low Resolution Aspect Ratios
Free Aspect
5:4
4:3
3:2
16:10
16:9
Standalone (1024x768)
1080o (1920x1080)
Add
Label
Type
Fixed Resolution
Width & Height
1920
1080
1920x1080
Cancel
OK
emaining
Turn Indicator
Turn: 0/15
Turn Timer
Project
Console
Create


Unity 2018.2.14f1 Personal (64bit) - GameScene.unity - CPS Game v2 - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component Window Help
Center
Local
Scene
Game
Display 1
1080o (1920x1080)
Scale
Maximize On Play
Mute Audio
Stats
Gizmos
Hierarchy
Create
GameScene
Main Camera
GameController
WaterFlowController
SceneLoader
Canvas
Directional Light
EventSystem
GameBoard
Plane
Attacks Remaining
0
Turn Indicator
Turn: 0/15
Turn Timer
Next Turn
Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
GameLogic
Attacker
Defender
Modules
Prefabs
Scripts
Scenes
Packages
Assets ▸ Scenes
GameScene
TitleScene
TutorialSce..
VictorySce..

Current Bug List and Missing Features

- Missing Features -

- No attacks on water tank
- Capacity of pipes and tank are not used
- Defender win screen does not show ratio of purity
- Still need to implement a WIN condition

- Bugs -

- Win screen for defender does not show water ratio properly (This would need to be properly adjusted anyway since we currently haven't included a WIN condition; The next group would need to devise a good way of appropriating points {Game should be POINT based as discussed with Dr. McMillin})
- Null exception error for avaliables attacks (due to scene changing and the object not being on screen)
- Tank cannot be attacked since we haven't fully devised a plan on how to "attack" or "defend" the tank, since the tank object contains a LIST of individual water objects

Suggestions of Improvement from Client

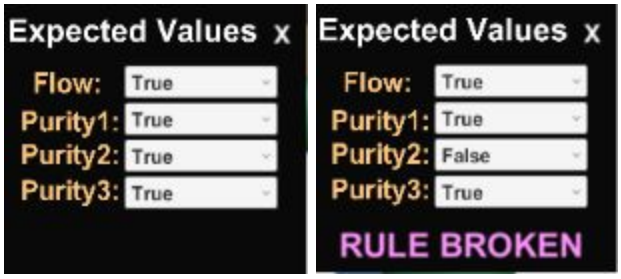
- Ability to destroy objects on map (ie. burst pipe, burst water tank…)
- Attacker can attack owls
- Pump has other variants beside on and off.
- Pump can have variation of flow. (ie. 50% open)
- Eye color of owl matches what he can see
    - ADA compliant
    - Take in consideration of color blind
- Have an object to represent the attacker and match with colors what he can see
- Graphs in 3D to show risk of attack for each component
- Win condition based on point system
    - Defender gets points for water in reservoir, cleanliness of water, successful defend
    - Attack gets points for water missing in reservoir, destroying a system component
- Help system when stuck
- Possibility of random map generation
- Attacker can shut down system when broken specified components
    - Purpose to show that CPS is about how long the defender can defend the system and that there will always be an attacker trying to destroy the system. The defender may be taken offline, but there will always be an attacker and he can never be destroyed.
- When sensor is attacked, it is only attacked for *x* many turns

- Make more invariants so that they system mimics a real CPS
- Make the defender logic compare two values rather than be a set of checks at specified points

High Level Technical Overview of some key points within the program

- Assets Folder holds all of the code logic that we built
    - Within the Assets folder we have 3 other folders:GameLogic,Modules,and Scenes
    - **Scenes** Generally work with how our screen notifies that it is the Start of the Game, End of the Game, Attacker's Turn, Defender's Turn, and other small changes outside of the actual game board
    - **GameLogic** generally manages the turns, be it the defender's turn or the attacker's turn. Think of it as a turn manager.
        - **GameController.cs** manages the turns and does some backend calculation depending on the previous turns, attacker and defender.
            - One can see that a function **TickModules()** is called, this function lets the game know that the flow of water should be moved down the pipes or prefabs in this case. When you go up to the programming hierarchy you could see that inside the **WaterFlowController.cs** within the **TickModules()** function, the **ReservoirTick()** function is the one that initiates the movement of water.
        - Defender Folder houses Owl Logic
            - **Oracle.cs** handles owl logic and calls upon Valuation.cs when using the Owl pointers. Applies the rule specified and checks whether or not the rule is broken, if the rule is broken, **Valuation.cs** will handle the problem. The owl also fixes the attack IF the attack is cornered by the owl with a set of 3 objects (i.e. Sensor1 Filter1 Sensor2). The **ApplyRule()** function looks at the 2 owl points, which is called by the **ModuleMatchesExpected()**, and compares the expected and actual values between the sensor/filter data and the water contained in the object.
            - **Valuation.cs** initializes the small popup that allows the user to specify desired rules set to follow and will prompt the user a "RULE BROKEN" at the bottom of the popup if the water object that passes doesn't have the same proper identifiers specified, however, it is a double edged sword as the defender can accidentally state that Filter 3 should hold 2, even though the owl

is only pointing at the sensor before Filter 3.



        - Attacker Folder just houses some of the prefabs used within gameplay
    - **Modules** houses the prefabs (i.e. objects on screen like a Filter) and scripts attached to those prefabs. These scripts act on or get triggered to initiate depending on the user's action during gameplay.
        - **Module.cs** handles the user actions on the game screen like clicking on a prefab during the attackers turn. flow of water between each prefab
        - Script Folder holds all the scripts that we attach to the prefabs to manage data/object management given a specific event handler.
- Key points to remember
    - **Water is ticked from the end of the line** (reservoir) **to the front**. Basically water is dumped into the reservoir from the last pipe, then it goes up the line up to the first pipe. This is to negate the necessity of adding a CopyWaterObject, as we would just need to bring the past WaterObject down the list.
    - Generally managing object creation and script attachment requires some programming knowledge of Unity, and learning the basics of prefab creation and script writing will be greatly useful before attempting to update or work with the given code. Good luck!