CS 3160 – Game Programming

Project 6 – Multiplayer

Learning Objectives

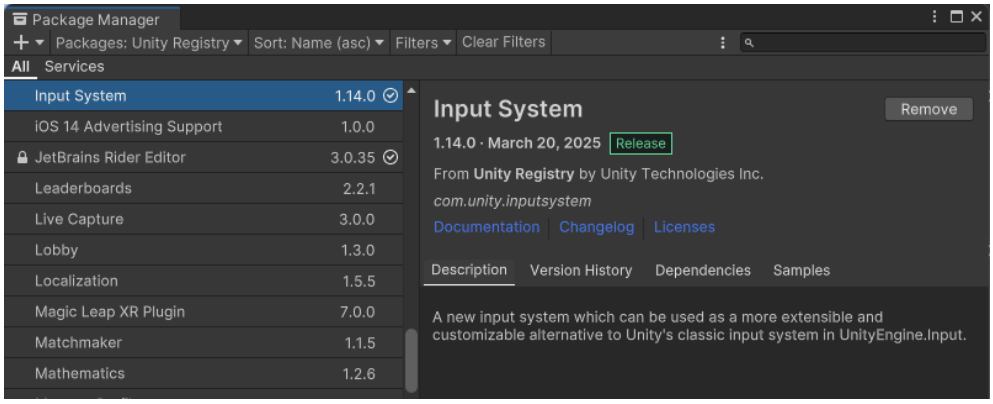
- Learn how to implement online multiplayer

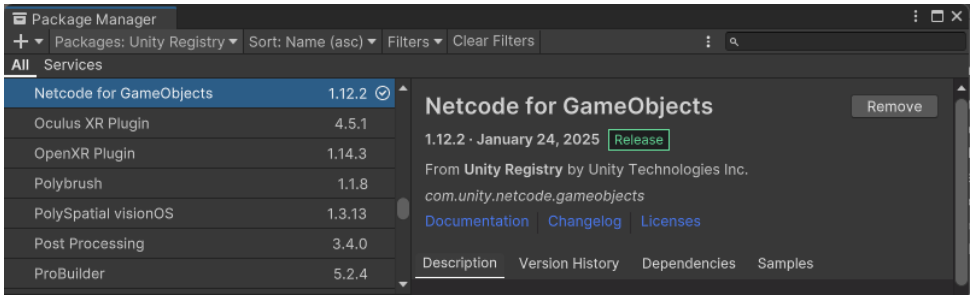
Overview

In this project you will become familiar with adding multiplayer functionality to your game.

Sample Code

To complete this project some sample code is required. This is provided in the form of a Unity Package. Download the "SampleCode-Project6" Unity Package from Pilot. First, create an empty Unity project using the "3D (Built-In Render Pipeline)" template. Then, go to "Assets" > "Import Package" > "Custom Package…" then locate "SampleCode-Project6.unitypackage" downloaded from Pilot and import all files. After this, you must install the "Input System" and "Netcode for GameObjects" packages provided by Unity here:





If importing was successful you should be able to load the "GameScene" Scene in the "Scenes" folder and you can test the game.

NOTE: Importing a Unity Package does not import Layers. The layers used are listed above.

Multiplayer

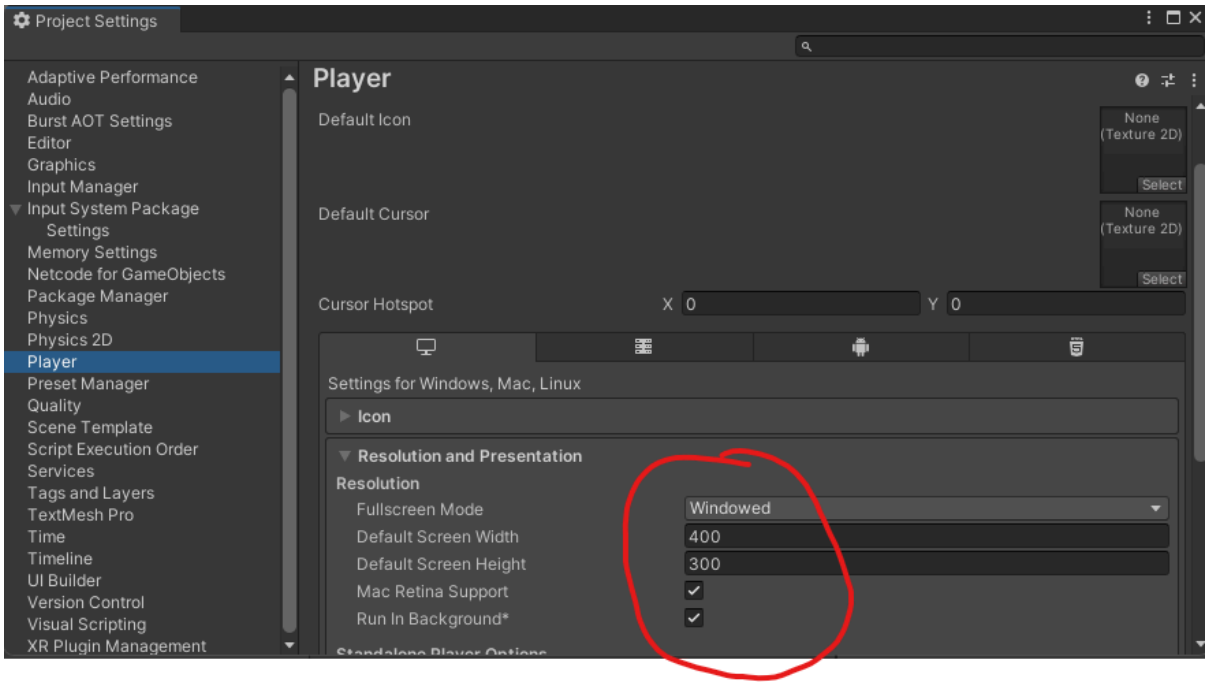
This multiplayer game is a snowball fight between cubes. All of the players in the game are a uniquely colored cube and can throw snowballs at other players. If a player is hit with a snowball they are eliminated from the game. Make the following modifications to the sample code to implement proper multiplayer functionality:

- Player Prefab
    - CubeController.cs:
        - private void Start()
            - If we are not the owner, disable the camera (prevents other player's cameras from overriding our camera's view)
            - Get an array of spawn points (type SpawnPoint) and spawn at one based on the player index
                - Hint: the player index is the player's number in the game. This can be found via playerInput.playerIndex.
            - Get an array of colors (type Color) and set the renderer's material's color to this color
                - Hint: can be set using GetComponent<Renderer>().material.color
                - Hint: colors can be a public array of Color that can be specified in the Inspector
        - private void Update()
            - Only allow controlling movement if we are the owner of this player (prevents us from controlling other player's positions)
        - private void FixedUpdate()
            - If the player is dead, lerp the transform's scale from its current scale to Vector3.zero.
        - private void RequestThrowServerRpc(int playerIndex)
            - The argument "playerIndex" specifies who threw the projectile
            - Is a [ServerRpc] executed by the server but called by the players
            - Instantiates a new projectile, spawns the network object in the server, sets a NetworkVariable<int> on the projectile to identify who created the projectile
                - Hint: To spawn a projectile on the server, you have to get the NetworkObject component and call the "Spawn()" method
                - Hint: To set a network variable's value you have to get the field "Value" and set it to equal to the desired integer
        - public void OnThrow()
            - Call a ServerRPC to request instantiating a new projectile
        - public void KillPlayerClientRPC(int playerIndex)
            - The argument "playerIndex" specifies which player dies
            - Is a [ClientRpc] executed by the clients but called by the server
            - Kills the player only if the playerIndex is this player's index
                - Hint: When the server sends this RPC to the players, every player will execute it, but we only want to kill the player specified by player index
            - Find an object of type "SpectatorCamera" and call "EnableCamera()" method
            - After a player has died, they should not be allowed to move or fire projectiles

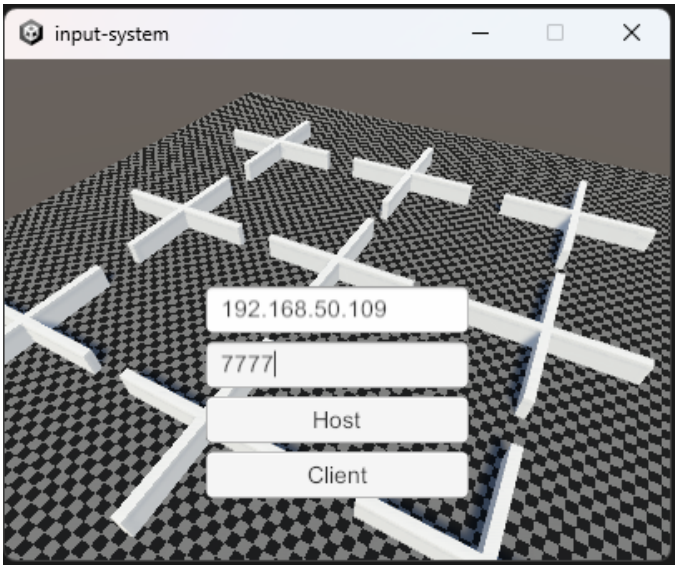
- Projectile Prefab:
    - private void OnCollisionEnter(Collision collision)
        - If this collision occurred on the host (i.e. if (IsHost)) then check if collided with a player (i.e. has a CubeController component) and check if the NetworkObject is spawned
        - If collided with a player and the playerIndex of the projectile is NOT the playerIndex of the player hit (the projectile hit a player other than the player that threw the projectile), call the KillPlayerClientRPC on the CubeController
        - Get the NetworkObject and Despawn() it

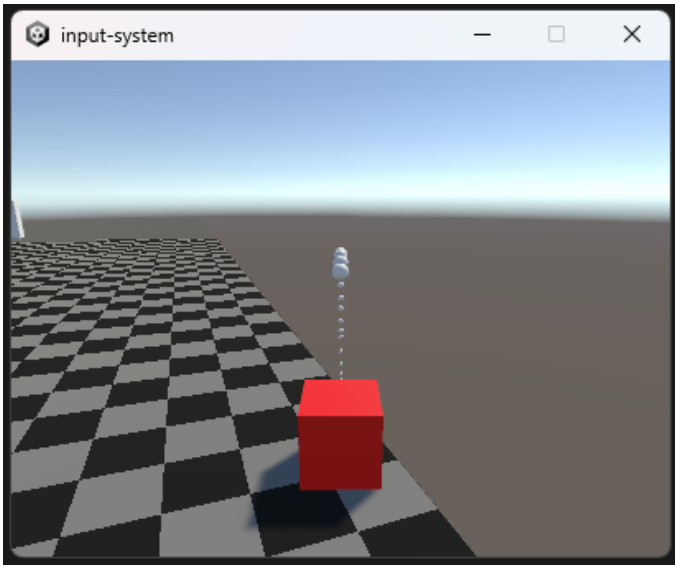
Note: You do not have to do error handling for if the IP address and port cannot be connected to.

Note: Netcode for GameObjects does not work with WebGL builds. Use a desktop build instead. To have multiple game windows running in a small resolution use the following project settings:

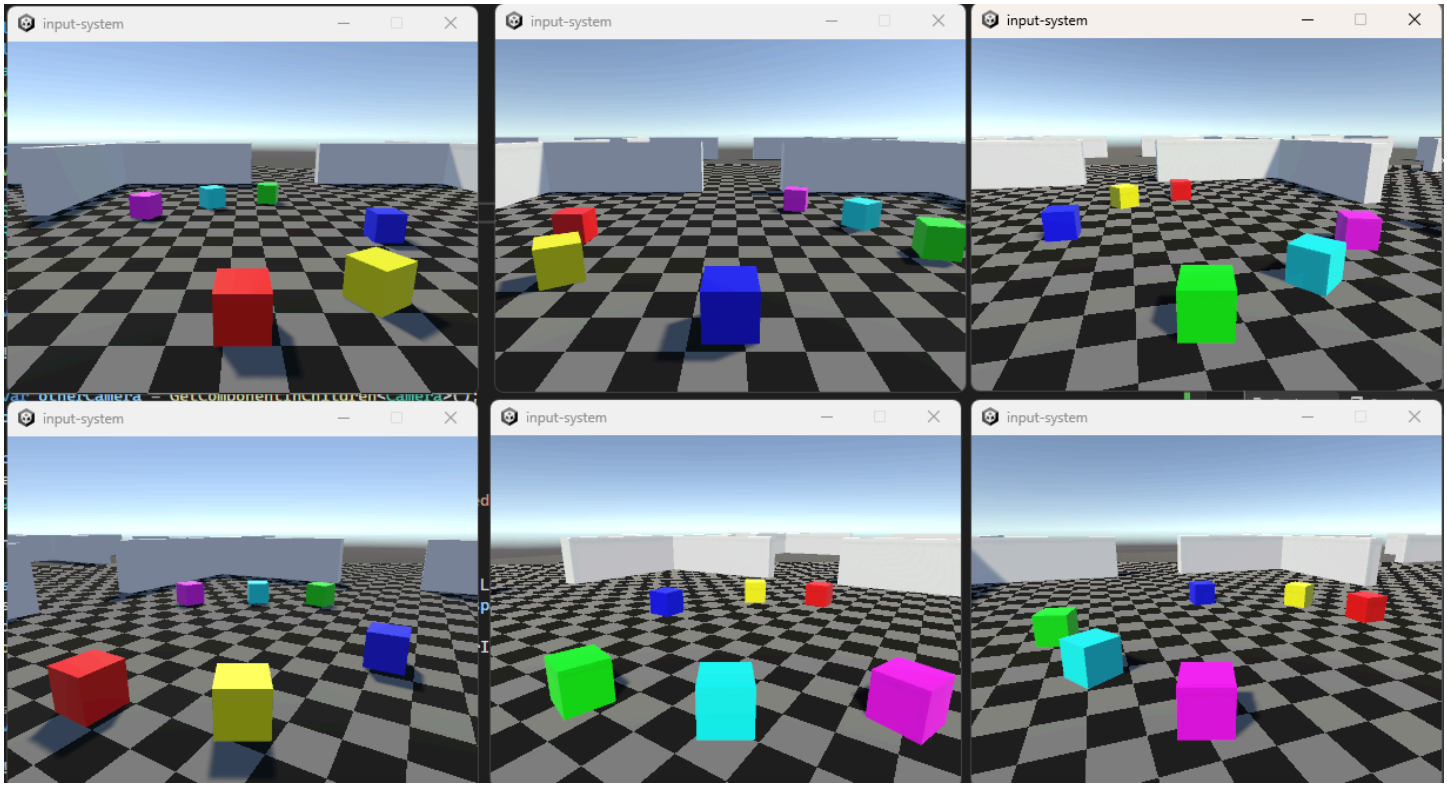


If everything is working correctly you should see this (connecting to a game and throwing projectiles):

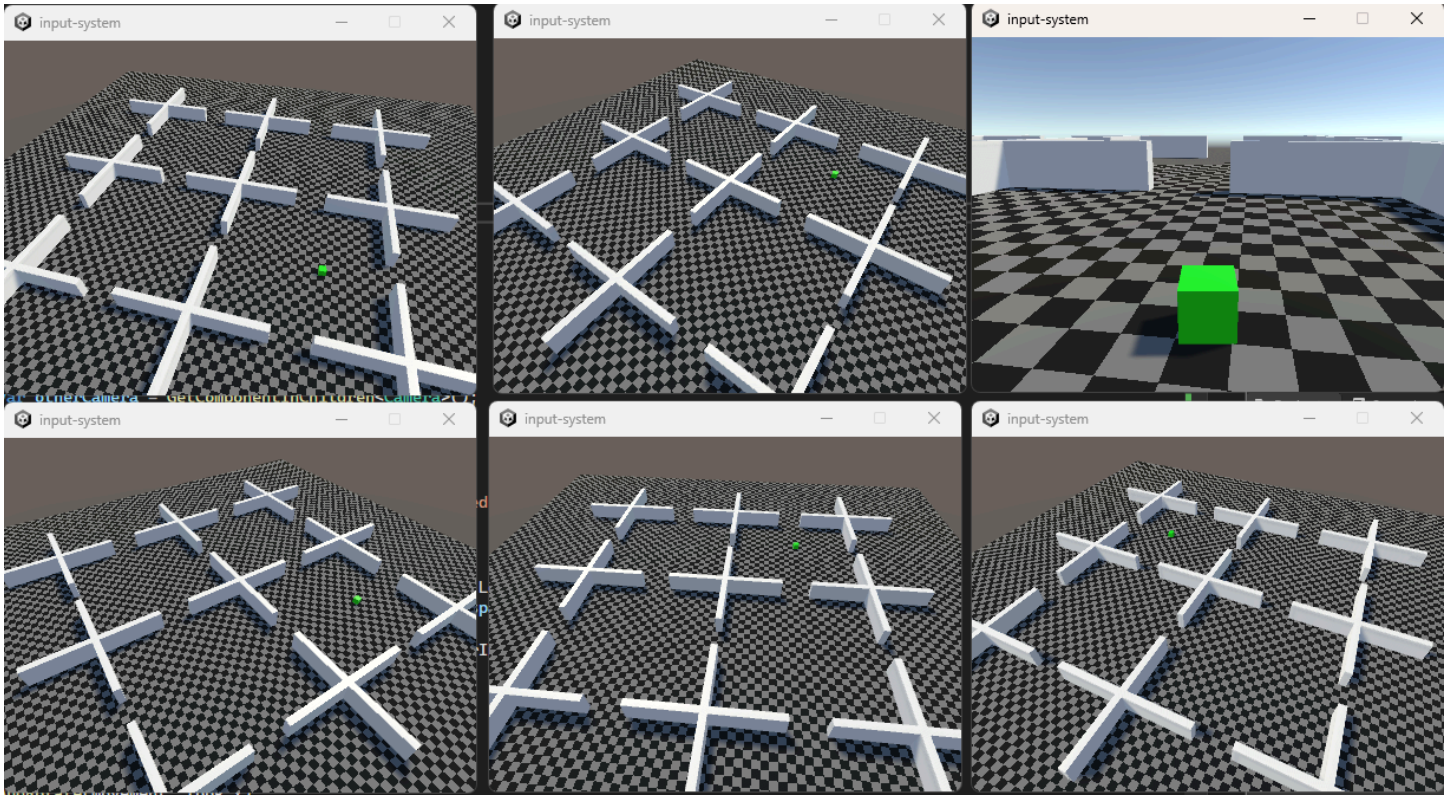




Multiple players in same game:

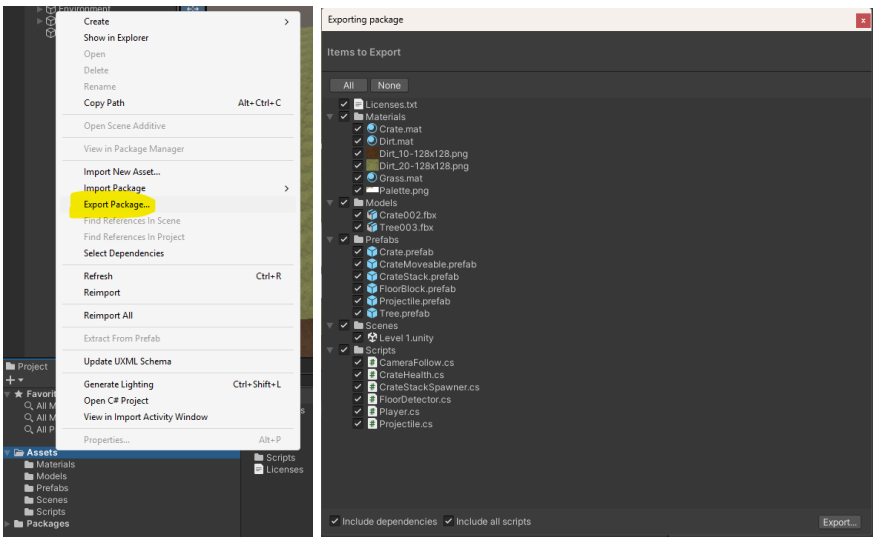


Green player eliminated all other players:



How to Submit

To submit your code, you must pack your project into a Unity Package. To create a Unity Package first right click on the “Assets” folder inside of the Unity Editor, select all items and hit Export. Lastly, name your file “YourLastName-Project6” where “YourLastName” is your actual last name, and submit this file to the Pilot dropbox:



Grading

This lab is worth 5.00 points, distributed as follows:

Task	Points
There are no build or runtime errors	1.00
Implemented CubeController colors and spawn points	1.00
Implemented CubeController ServerRpc and ClientRpc	1.00
Implemented CubeController dying	1.00
Implemented network Projectile collisions	1.00
Total	5.00