# Better Tower Defense - Requirements

Jacob Dufault

## Entity System

See the spec for a detailed description of why the entity system is interesting. See the design document for an overview of it. Here is an example of usage, and what the required output is, versus incorrect output.

The following example implements behavior for all game objects which have “TemporaryData”, ie, they only want to exist for a given number of updates.

```
public class TemporarySystem : SystemBehaviour {
    [InvokeOnStateChange]
    [FilterAddedData(typeof(TemporaryData))]
    public void InitializeTemporaryData(IEntity entity) {
        Log.Info("Created {0}", entity);
        TemporaryData temp = entity.Current<TemporaryData>();
        temp.Modify().UpdatesLeft = temp.UpdatesLeft;
    }

    [InvokeOnUpdate]
    [FilterModifableData(typeof(TemporaryData))]
    public void DecrementTemporaryDataCounter(IEntity entity) {
        TemporaryData temp = entity.Current<TemporaryData>();

        Log.Warning("ModifyUpdate {0} with {1}");
        temp.Modify().UpdatesLeft--;

        if (temp.UpdatesLeft == 0) {
            entity.Destroy();
        }
    }

    [InvokeOnStateChange]
    [FilterRemovedData(typeof(TemporaryData))]
    public void PrintRemoved(IEntity entity) {
        Log.Info("Removed {0}", entity);
    }
}
```

Driving code to create the entity which will be used in the above system:

```
GameObject go = new GameObject();
TemporaryProvider temp = go.AddComponent<TemporaryProvider>();
temp.Initialize(new TemporaryData() {
    UpdatesAlive = 3
});
```

Of particular note, and one aspect of what makes this entity system approach novel, is the complete separation of role; the driving code only cares about *data*, while the TemporarySystem only cares about *business logic*.

Output from the driving code:

```
1: Created Entity [uid=1]
```

```
2: ModifyUpdate Entity [uid=1] with TemporaryData [UpdatesLeft=3, UpdatesAlive=3]
3: ModifyUpdate Entity [uid=1] with TemporaryData [UpdatesLeft=2, UpdatesAlive=3]
4: ModifyUpdate Entity [uid=1] with TemporaryData [UpdatesLeft=1, UpdatesAlive=3]
5: ModifyUpdate Entity [uid=1] with TemporaryData [UpdatesLeft=0, UpdatesAlive=3]
6: Removed Entity [uid=1]
```

The entity system is obviously significantly more complicated than this (for example, how data is specified, instantiated, and destroyed), but the previous gives a sampling of correct usage and output. Incorrect output would be, for example, if lines 6 and 1 in the output were swapped.

## Networking

The networking model supports a number of interesting features. Correct output is being able to connect to an arbitrary IP address that does not require port forwarding (though some IPs that require port forwarding are supported due to support for NAT punch through).

Incorrect output would be being unable to connect to an IP.

## Unit Spawning

Unit spawning involves creating new objects/enemies at runtime in a predetermined order. Correct output is doing this as specified by the designer; incorrect output it spawning enemies at the wrong time, in the wrong order, or the wrong quantity.

## Locomotion

If path-finding is locating a path between two nodes in a graph, then locomotion is the process of navigating between those two nodes.

Correct locomotion involves successfully following a path and not, for example, running through multiple enemies and ignoring effects that are applied to the object.

## Effects

Effects are applied to objects within the game that somehow modify the state of the object; the innovative feature being that effects are dependent upon other effects.

Correct effect application involves correct detect of effect combinations and applying some modification to the unit being affected. Incorrect effect application would be invalid modifications being applied.

## Building Placement

Building placement involves the placing of buildings around the map. That is, placing buildings in locations where they can attack monsters. Correct building placement ensures that buildings cannot be placed in invalid locations; that buildings, when placed, are correctly hooked up to all necessary game systems, etc… Invalid building placement would be, for example, if the player could place buildings such that it made it impossible for the user to lose points, or a building, when placed, did not correctly inject itself into the spatial optimization systems (ie, the various quadtrees).

## Resources

Resources are used for building new towers/buildings. Correct resource behavior ensures that the player does not go below some minimum resource limit, that building buildings successfully reduces the players resource count, and that resources can be gained. Incorrect behavior would be if a player could gain an infinite number of resources.

## Power System

The power system allows a player to place arbitrary effects onto regions of the map or onto specific units. Correct behavior involves insuring that the cost of the power is deduced and that the effect is correctly applied to the region or target. Incorrect behavior would be the opposite as described above.

## Level Types

Different level types impact how units are spawned, who the player is fighting against, among other details. Correct behavior depends on the current level type, but correct behavior requires that the level type operate consistently, allows a player to win, and allows a player to fight against an enemy. Incorrect behavior would be, for example, if one level type suddenly changed to another level type.

Level types are, overall, a collection of different game-level abstractions which tie together in a novel way; the way these abstractions tie together defines the level type.

## Dynamic Difficulty

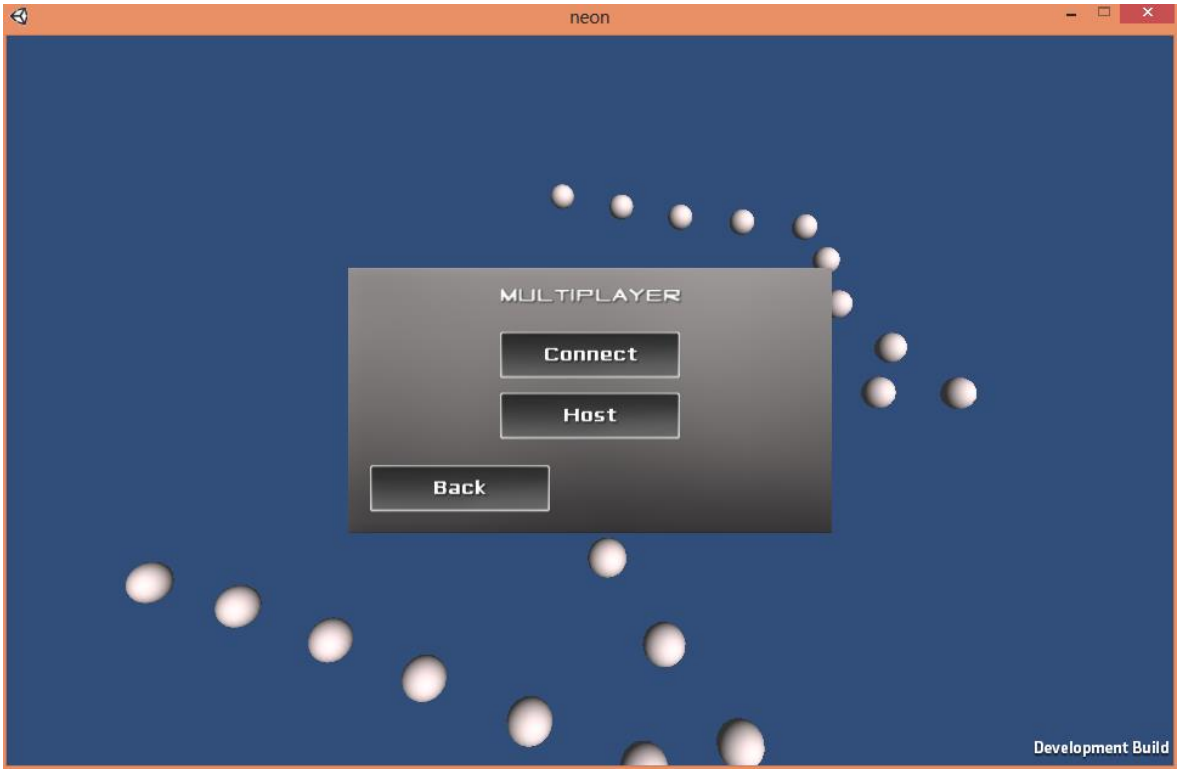
Dynamic difficulty requires that the game increase difficulty (such as by spawning more enemies or increasing tower cost), however, the game cannot decrease difficulty beyond some base minimum to prevent abuse. Incorrect behavior would be allowing the player to abuse some basic loophole in the system; such as by not building any buildings and having the difficulty scale to zero.

## GUI mock-ups

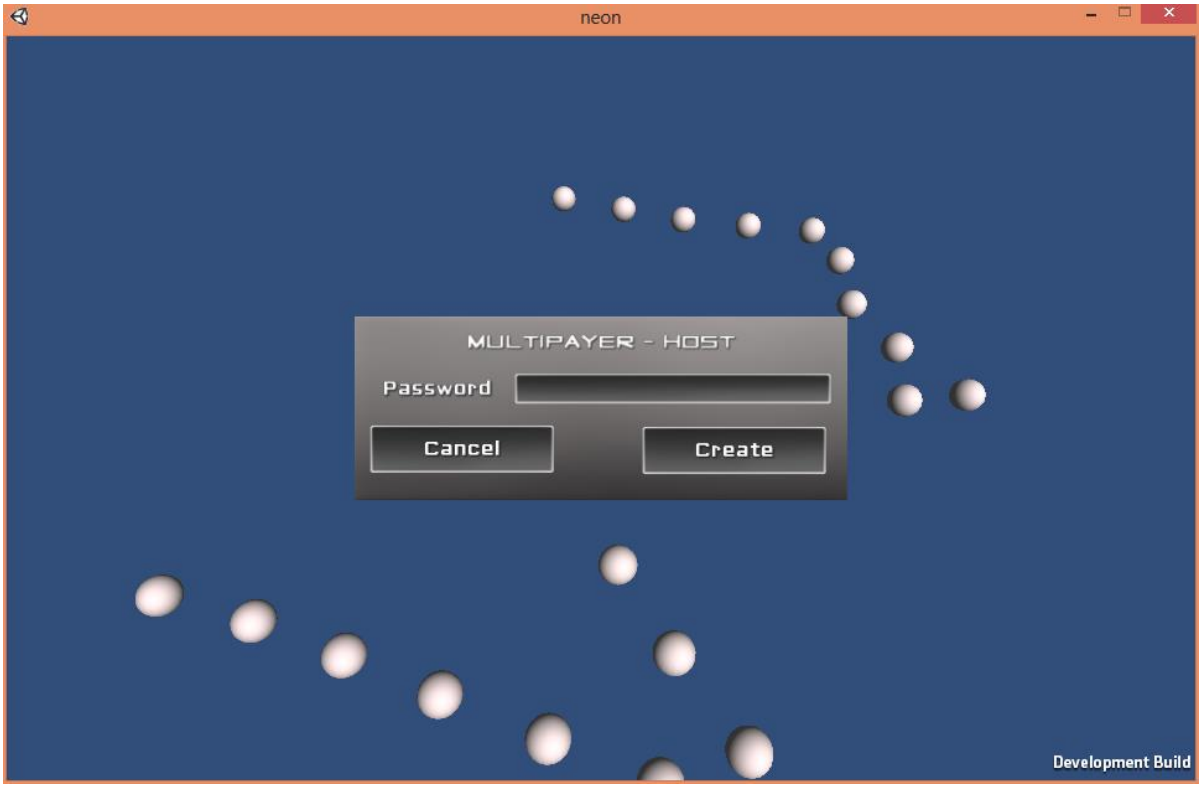
Here are some of the UI mockups. There is more UI work to be done, and the current UI is very early, but the basic details are there.

Main menu

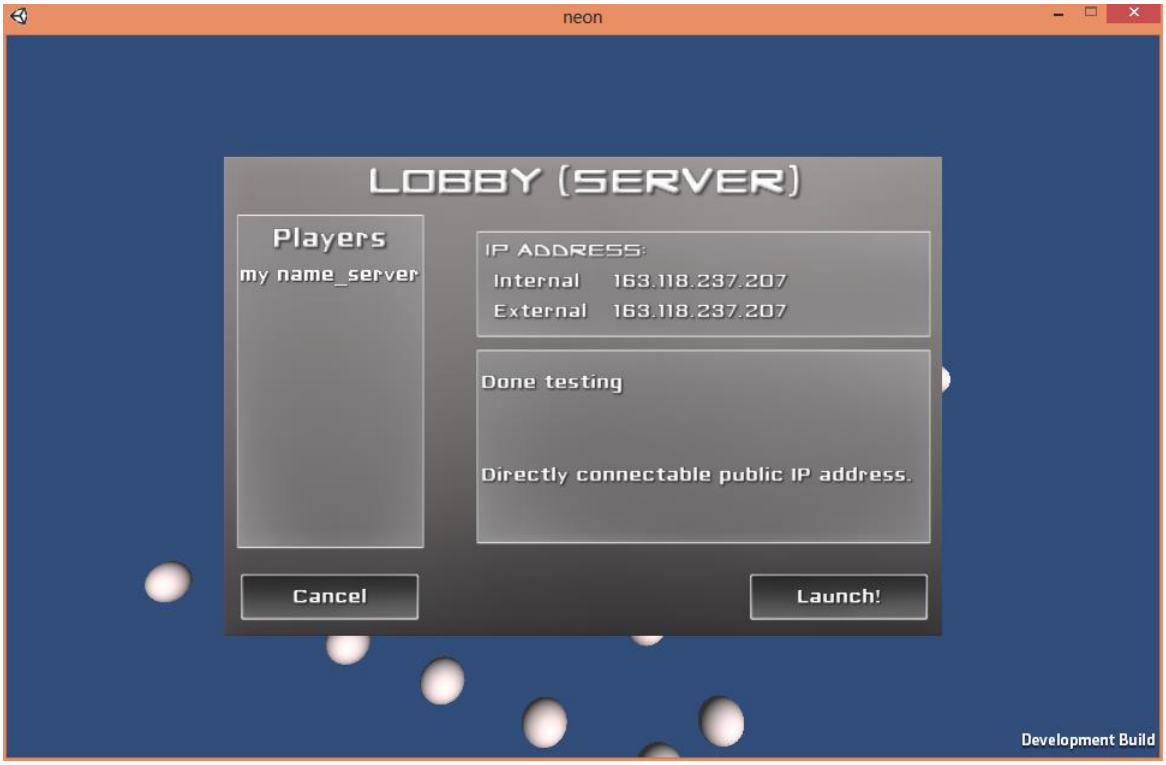
Multiplayer menu



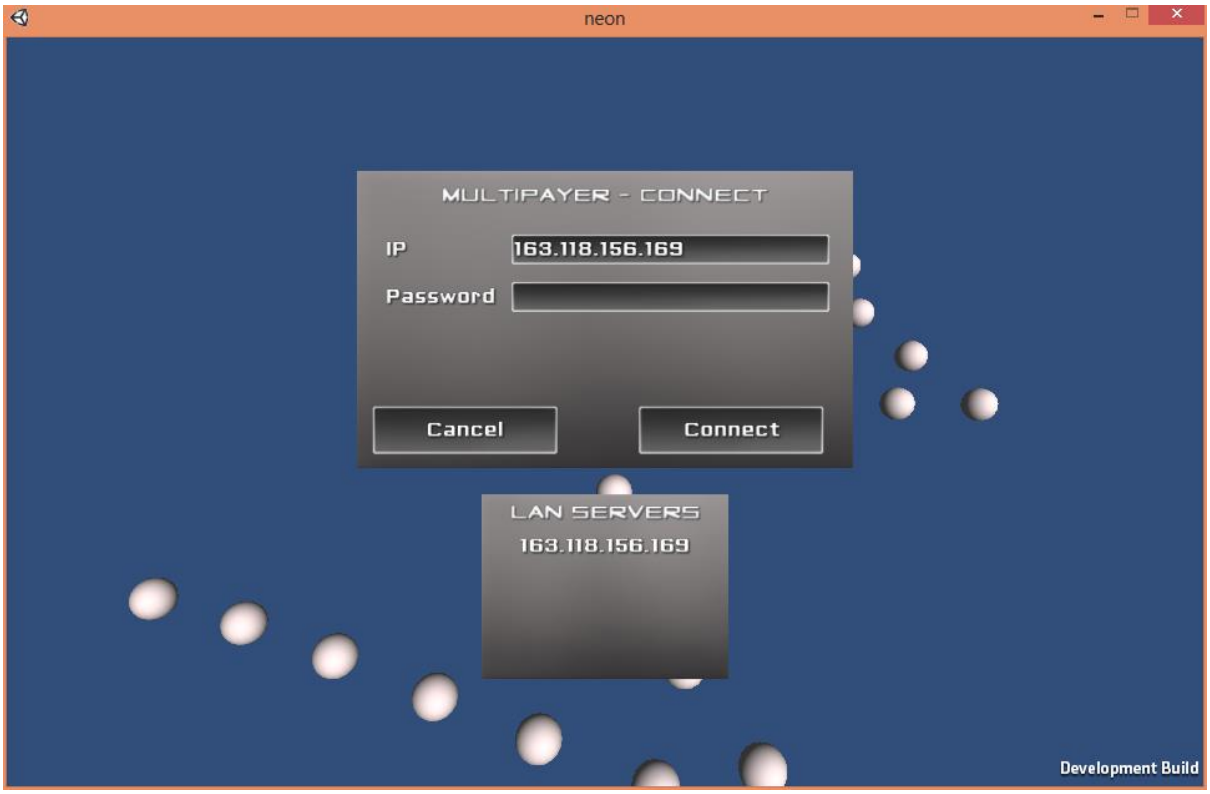

Multiplayer setup panel



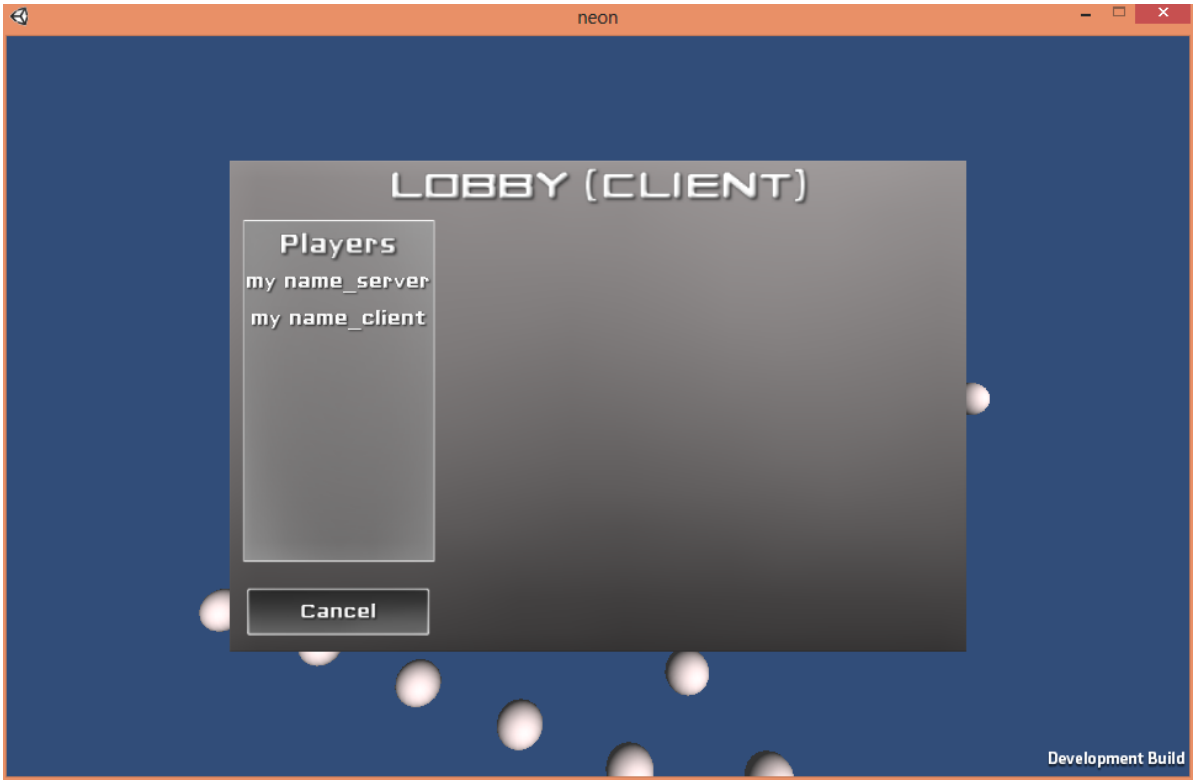
Lobby for hosting a server



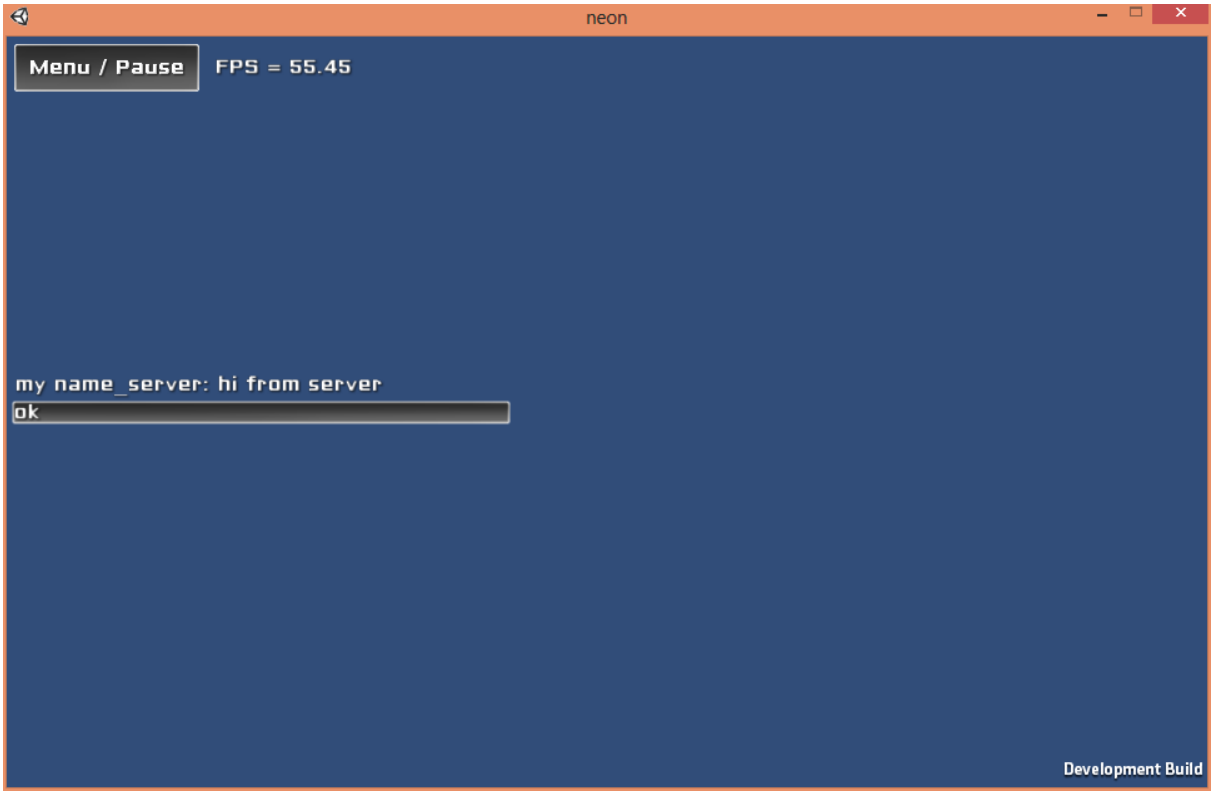

Connecting to a server (with LAN host discovery)



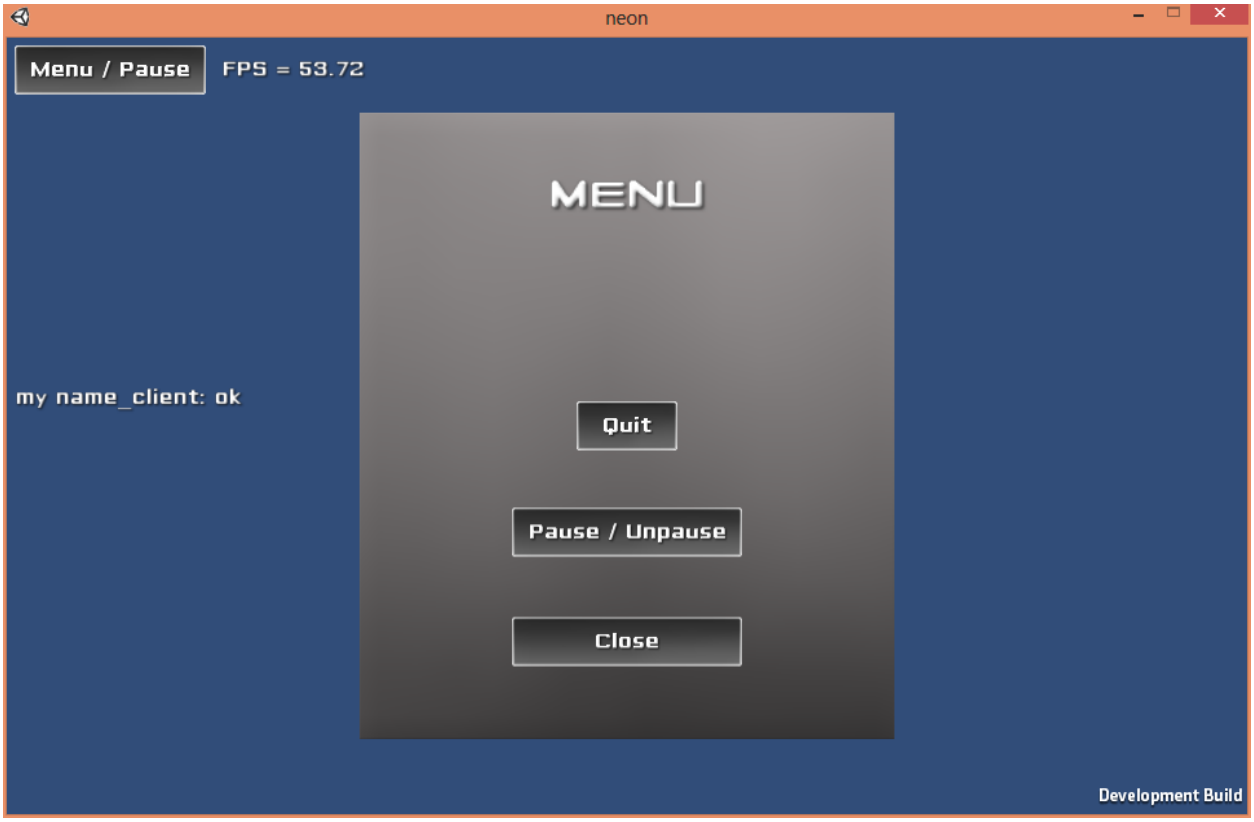
Lobby client interface




Chat interface



In-game menu




In-game GUI mockup