John Lemon's Haunted Jaunt

From Unity to PlayCanvas

By Christina Kaliora

About me

- Hello! I’m Christina!
- I’m a Games Programmer from Greece and I first opened PlayCanvas in September of 2016 😊
- Self Studied programming -> Coding Bootcamp -> Game Development
- Solar Games

About the Project

- John Lemon is a Unity tutorial project.
- I chose it because it was a fun game prototype with various different features.
- Environment and assets were fully provided and the art style was cool!

A great challenge to see that game running in a browser!

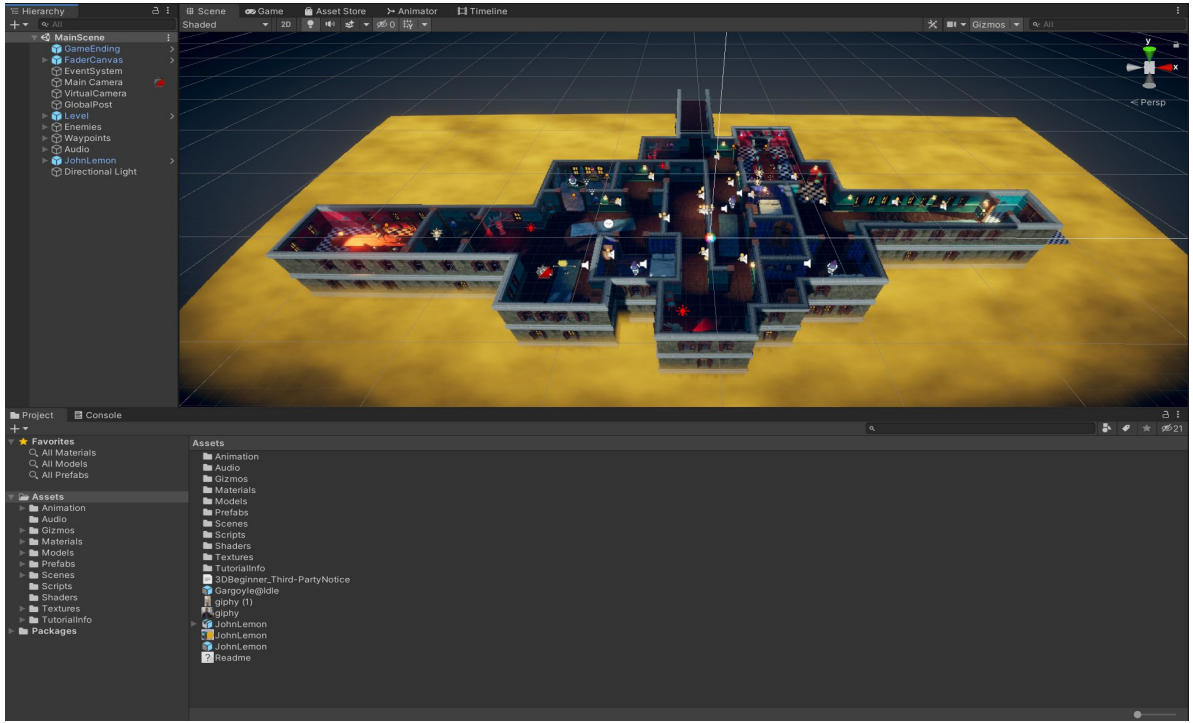
Why PlayCanvas Over Unity?

- PlayCanvas editor is web based!
- You can collaborate in real-time with multiple users
- Unity’s WebGL export doesn’t work in mobile devices
- PlayCanvas fully supports all devices that can run a web browser
- PlayCanvas games can even run as native applications (Electron!)

From Unity to PlayCanvas

- Overall a smooth experience doing the conversion to PlayCanvas.
- All assets could be directly uploaded to the PlayCanvas editor.
- In this presentation I focus on key points that presented some challenges.

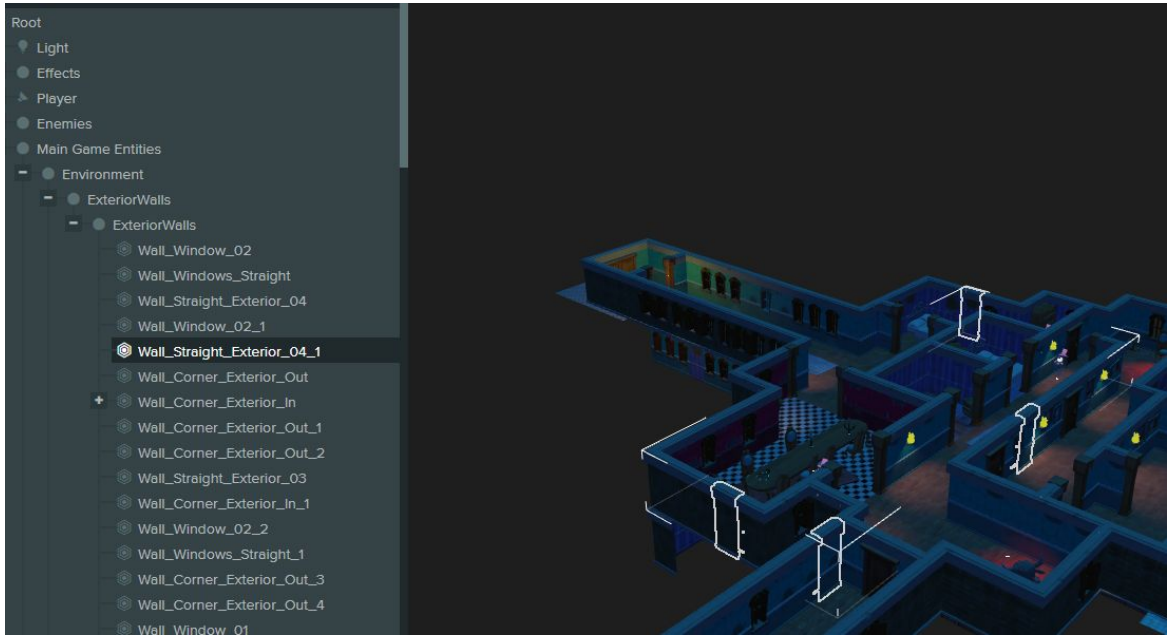
From Unity to PlayCanvas



Transferring The Game Environment Assets

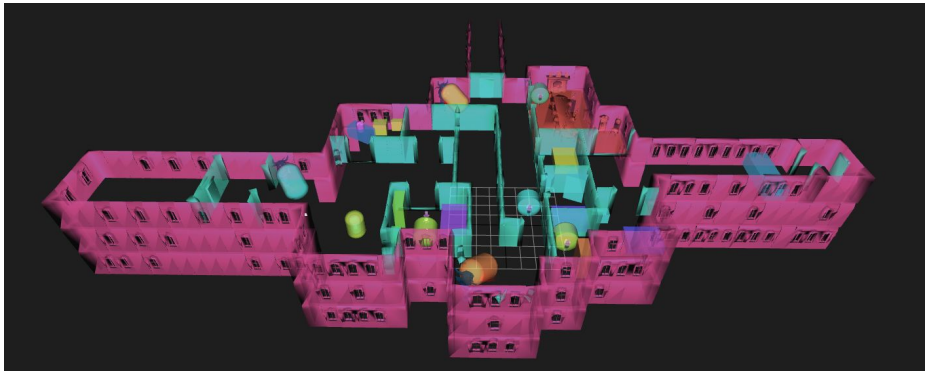
- The game used modular pieces for the walls and floor.
- To avoid placing each and every part manually, I used the Unity FBX Exporter to export the full level geometry to FBX.
- Importing that FBX to PlayCanvas added the hierarchy to the scene while using those modular pieces.

Transferring The Game Environment Assets

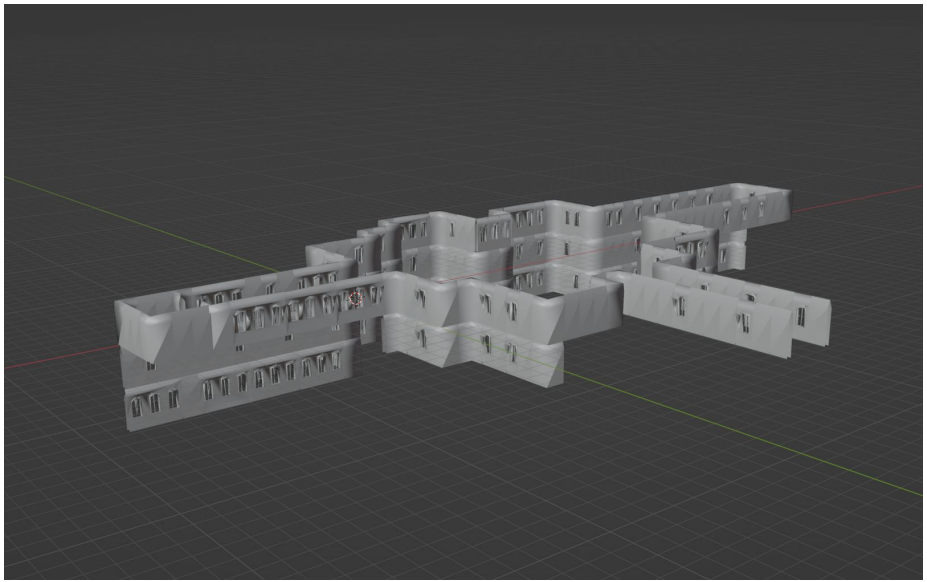


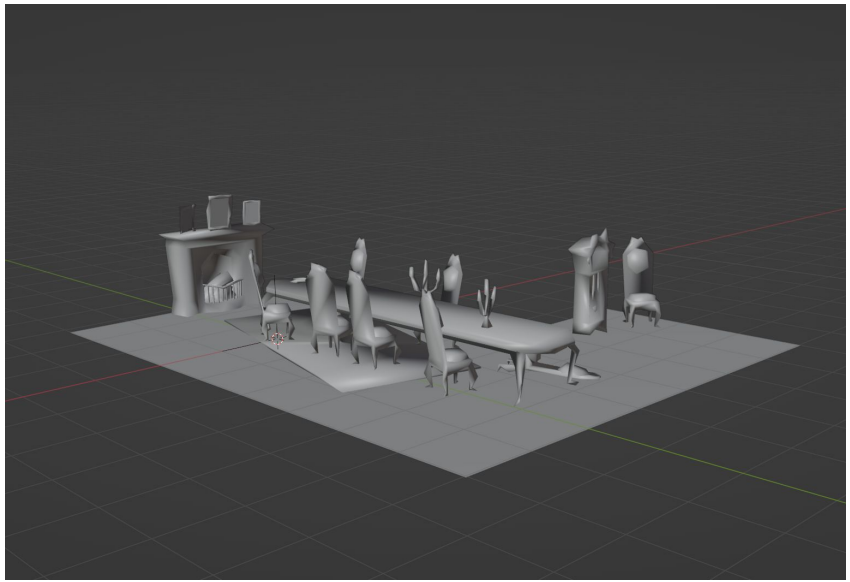
Adding Colliders easily

- The models are now in place but given the huge number of entities it will take a long time adding colliders to each.
- I solved this quickly by transferring the models in Blender, joining them and decimating to get very light trimesh colliders.



Adding Colliders easily



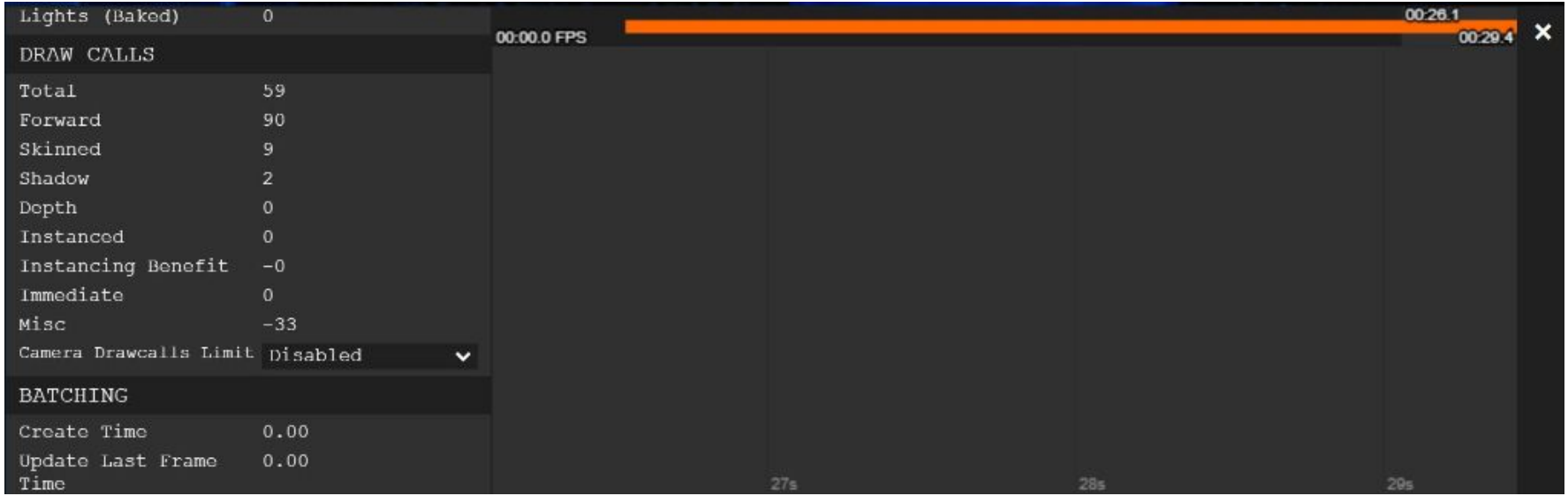


Improving Performance

- Right now the game had too many draw calls!
- I could use PlayCanvas batch groups, but I had the privilege to be in the Uranus Tools for PlayCanvas beta.
- I simply added the Uranus Instancer to the project and draw calls were reduced to more than 50%.
- How Uranus Instancer script works?

Improving Performance - Before

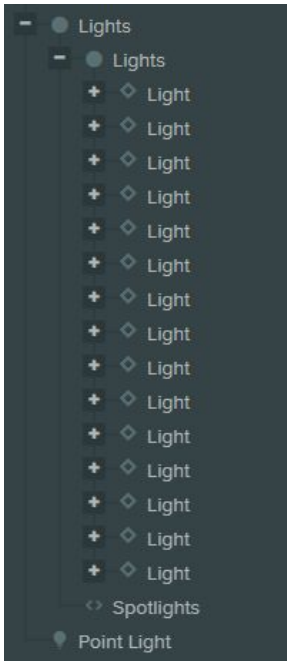
Improving Performance - Before

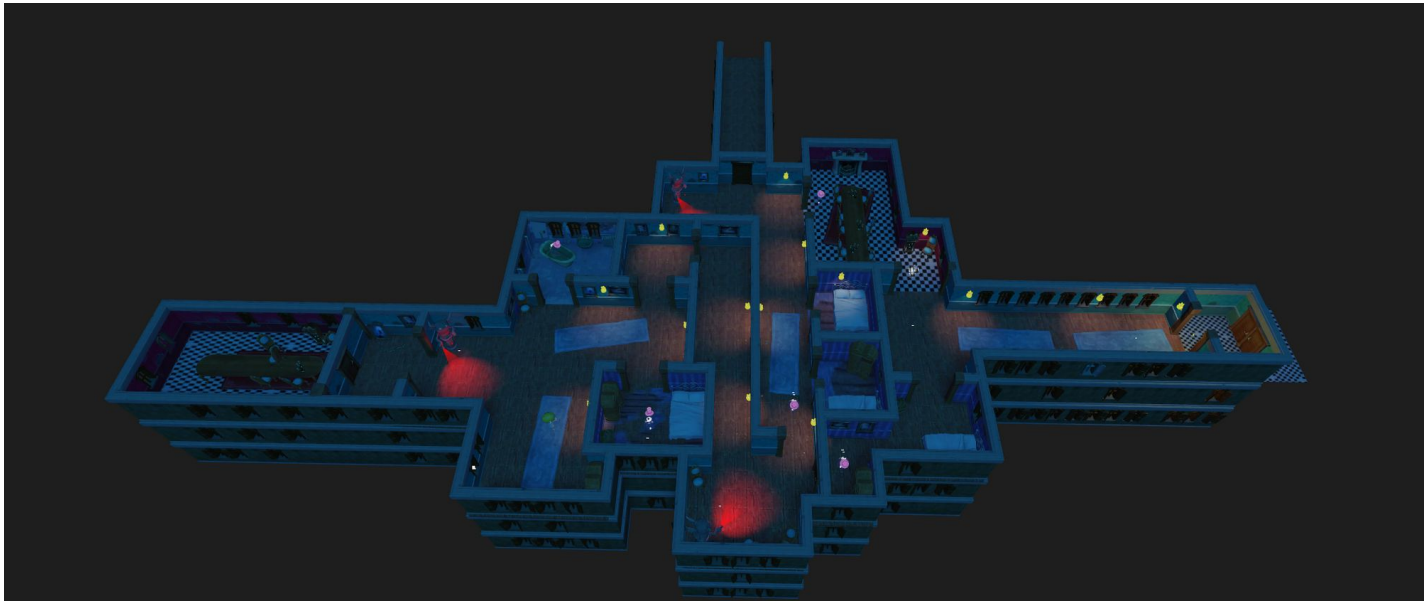


Improving Performance

- The scene has 19 dynamic lights in total.
- In some places as many as 10 lights can be visible at the same time, that had a big performance hit!
- Luckily PlayCanvas **clustered lights** were available to easily add in the game. That solved this problem fully!

Improving Performance





Enemy Pathfinding

- I needed a quick solution for pathfinding navigation for the ghost enemies.
- Using waypoints was ideal!
- I took the “Camera following a path” example project and I was able easily to make my ghosts follow their paths.

Enemy Pathfinding

Ghost Rim Lighting Effect

- I used a custom shader by overriding the material **emissive** and **opacity** shader chunks.
- Using a fresnel equation based on the edge position and normal to increase lighting on the ghost edges.
- At the edge emissive and opacity factors are increased.

Ghost Rim Effect

Ghost Rim Effect - opacity chunk

```
RimLighting.opacityPS = `

float dFresnel;

#ifdef MAPFLOAT
uniform float material_opacity;
#endif

#ifdef MAPTEXTURE
uniform sampler2D texture_opacityMap;
#endif

void getOpacity() {
    dAlpha = 1.0;

    #ifdef MAPFLOAT
    dAlpha *= material_opacity;
    #endif

    #ifdef MAPTEXTURE
    dAlpha *= texture2D(texture_opacityMap, $UV, textureBias).$CH;
    #endif

    #ifdef MAPVERTEX
    dAlpha *= clamp(vVertexColor.$VC, 0.0, 1.0);
    #endif

    vec3 V = normalize(vPositionW.xyz - view_position.xyz);
    vec3 N = normalize(dVertexNormalW);

    dFresnel = 0.0 + 1.0 * pow(1.0 + dot(V, N), 3.0);

    dAlpha += dFresnel;
}
`;
```

```
vec3 V = normalize(vPositionW.xyz - view_position.xyz);
vec3 N = normalize(dVertexNormalW);

dFresnel = 0.0 + 1.0 * pow(1.0 + dot(V, N), 3.0);

dAlpha += dFresnel;
```

Ghost Rim Effect - emissive chunk

```
RimLighting.emissivePS = `

#ifdef MAPCOLOR
uniform vec3 material_emissive;
#endif

#ifdef MAPFLOAT
uniform float material_emissiveIntensity;
#endif

#ifdef MAPTEXTURE
uniform sampler2D texture_emissiveMap;
#endif

void getEmission() {
    dEmission = vec3(1.0);

    #ifdef MAPFLOAT
    dEmission *= material_emissiveIntensity;
    #endif

    #ifdef MAPCOLOR
    dEmission *= material_emissive;
    #endif

    #ifdef MAPTEXTURE
    vec4 emissiveSampler = $texture2DSAMPLE(texture_emissiveMap, $UV, textureBias);
    dEmission *= emissiveSampler.$CH;
    #endif

    #ifdef MAPVERTEX
    dEmission *= gammaCorrectInput(saturate(vVertexColor.$VC));
    #endif

    float rim = dFresnel * (1.0 - emissiveSampler.a) * 1.5;

    dEmission += rim;
}
`;
```

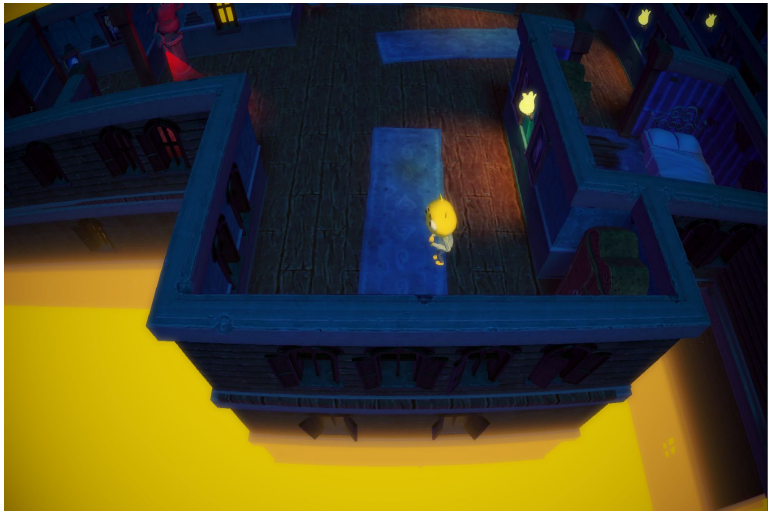
```
float rim = dFresnel * (1.0 - emissiveSampler.a) * 1.5;

dEmission += rim;
```

Camera Distortion Post Effect

- To give a haunted house perspective a post effect distorting the camera view.
- The effect distorts proportionally the screen pixels closer to the display edge.

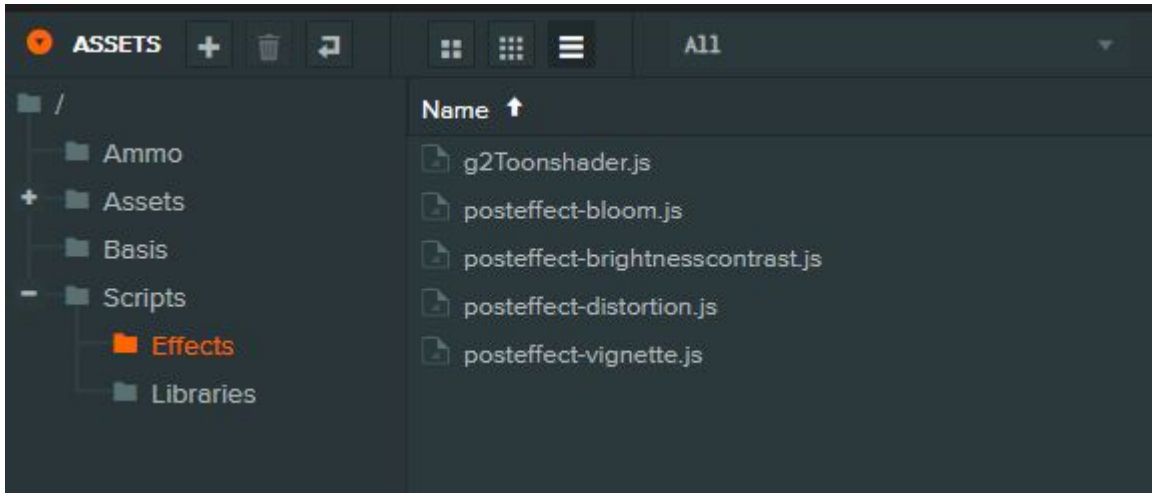
Height Based Fog Effect

- Using one of the Uranus Tools for PlayCanvas effects I was able to add a spooky height based fog surrounding the building.
- That effect works by overriding a global shader chunk and affecting the fog method of all scene materials.



Where to find the effects

You can find all the effects inside Scripts -> Effects folder.



Conclusion

- The development process was faster in PlayCanvas vs Unity.

- PlayCanvas download size and load time are much better compared to Unity, great for casual games and online experiences!

Thank You!

- **Play** the game: https://playcanv.as/p/atVPbI8K/
- **Public Project**: https://playcanvas.com/project/917469/overview/john-lemon-public-project
- **Twitter**: @christinaKlra
- **Portfolio**: https://solargames.io/christina/
- Take a look at **Uranus Tools for PlayCanvas**: https://solargames.io/
- Get started with **PlayCanvas** now! https://playcanvas.com/