Overcraft: ACG Final Report

Yuyang Wu*
Tsinghua IIIS
2023010871
yy-wu23@mails.tsinghua.edu.cn

Figure 1: The poster of OverCraft.

ACM Reference Format:
Yuyang Wu. 2018. Overcraft: ACG Final Report. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

We have successfully designed the game OverCraft, a 3D multi-player experience where players form teams, create rooms, and tackle various levels. The objective is to complete tasks such as cutting, frying, and plating dishes based on incoming orders, aiming to produce as many dishes as possible within the time limit. Players can communicate via a chat box, change their names, and customize their avatars by switching skins.

*Both authors contributed equally to this research.

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

2 Method

2.1 Technology stack

Our technology stack includes pure TypeScript for both server and client development. We use Three.js [8] for rendering, Ammo.js [5] for physics simulation, React [6] for UI design, Vite [10] for the front-end, and Node.js [2] for the back-end.

2.2 Entity Component System (ECS)

We utilize an Entity Component System (ECS) to manage data within our game. In ECS, data is organized into entities, such as a rigid body or player account. Each entity contains components, which are plain data types like Vector3, Geometry, or Material. Entities have no types, whereas components have specific types,

and each entity can possess at most one component of a given type. Systems operate by collecting entities with specific components.

This approach is particularly effective for online network games, as network-transmitted information requires a type tag to indicate its nature. ECS addresses this by transmitting components, which inherently include type information.

2.3 State Synchronization

The server does all stuff on its end, e.g. simulates a world with dynamic rigid bodies, and clients receives the data in each frame and render. For example, a client sends keyboard inputs to the server, and server receives the inputs, computes positions and rotations for all rigid bodies, and then send the data to all clients.

This approach is easy to implement and minimizes the risk of cheating. However, this method can increase the server's computational load.

2.4 Open Source Assets

We directly download assets from Sketchfab [9] or design models in Minecraft [7] and export them by jMc2Obj [4] and Blender [1].

3 Game Design

3.1 Player Control

It's better to modify player's velocity but not directly change it's position, since we use a physics solver and directly changing position can lead to issues like 3D clipping. When the player is moving horizontally, we set $v := v + \alpha\Delta t$, but here's a damping $d = 1 - \beta\Delta t$, in each frame $v := (1 - d)^{\Delta t} v$. When the player is not on ground, α is smaller and d is multiplied by a small ratio. This approach can make the movement smooth and the speed quickly increases to a certain value that is not related to Δt, although Δt can change due to the uncertainty of simulation time.

3.2 Catch item

We implement three types of catching.

- The first one use the `btPoint2PointConstraint` API in ammo.js to build a constraint, which is a constraint for the ODE solver. It forces two objects to be connected by a fixed length stick, and we change the rotation of the stick to be the rotation of mouse pointer every frame, so it presents a grabbing effect.
- The second one set the speed of the catching item to be proportional to the distance from the catching item to the mouse pointer, and the direction is to the mouse pointer, so the catching item goes to the mouse pointer quickly and smoothly.
- The third one deletes the catching item from the physics world and set a mesh at a specific relative position to the player, so it looks like the catching item is being held in the right hand.

3.3 Dishes

Each dish is a list of basic ingredients, and when the list matches the synthesis table, synthesis can proceed. When it is rendered, each ingredient is like a cylinder so that we can lay them vertically, and we set the collision box to be one cylinder.

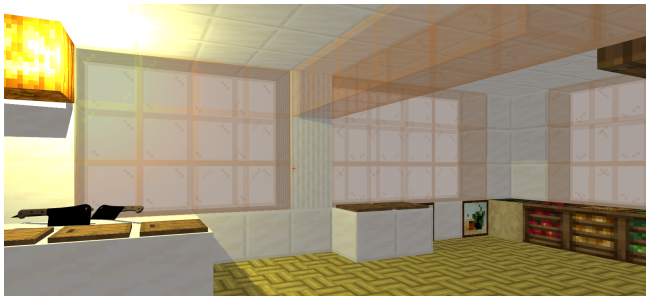

Figure 2: Level 2.

Figure 3: Level 3.

4 Results

We successfully design the game OverCraft. In OverCraft, players form teams, create rooms, and tackle various levels. They complete tasks such as cutting, frying, and plating dishes according to incoming orders to make as many dishes as possible within the time limit. Players can communicate through the chat box, change their names, or switch skins.

It achieves the basic requirements and these extra points in the Project Announcement:

Graphics Assets:

- Environment lighting (1pt)
- Syncrhonized audio (1pt)

Animation:

- Articulated objects (up to 3pts)

UI Design:

- Additional auxiliary interfaces (up to 2pts)
- User-friendly layout with visually appealing design (1pt)

Other Advanced Game Features:

- Mutli-player mode (up to 3pts)
- Customizable character appearance (up to 2pts)
- Increasing difficulty levels throughout the game (up to 2pts)
- Online multiplayer system (up to 4pts)

Completion:

- Easy installation or online access (up to 2pts)

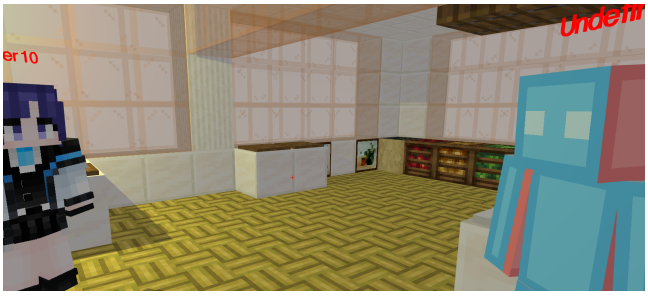

Figure 4: Multiplayer.

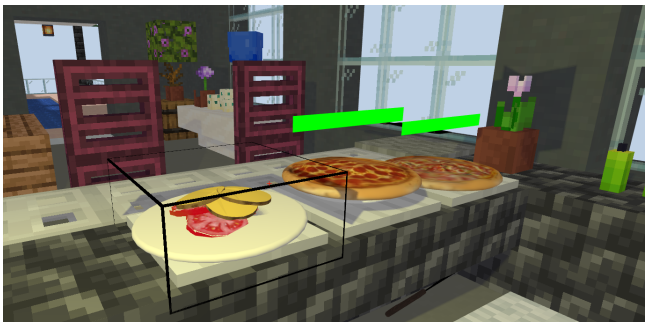

Figure 5: Various dishes.

5 Discussion

5.1 Code Structure

In our ECS implementation, not only Vector3, Geometry, Material, those things are implemented as components, things that need to be transmitted in network communication is also implemented as components, since entities must be set of components, e.g. `PlayerCatch` (describes the catch of the player), `PlayerKeyboardInput` (describes the keyboard input of the player). So for a new kind of message, we should create a new class and register it into the class table. Thus the code becomes more difficult to modify as more details of the game is implemented.

I think we need a more concise way to do internet communication, for example, we just use a huge JSON as the interface, and every modification is nicely handled. We should use fewer kinds of components.

5.2 Physics

Ammo.js causes headaches. Since it is directly translated from Bullet Physics (A Physics SDK implemented by C++) by Emscripten, we should manually free up the space of the variables from Ammo.js. Also, the time bottleneck is on Ammo.js, but we don't need such a physics engine since we only need a Minecraft background. So a better way is to use another lighter physics engine or design by ourselves.

5.3 Minecraft Background

We directly transmit the Minecraft model to a GLTF [3] model, which leads to a issue that we cannot include a large map into our game, because the geometry is very huge.

Since we use Minecraft, a better way is to analyze the architecture of the Minecraft saves and write a Typescript version to include the map.

6 Personal Contribution Statement

Other Advanced Game Features:

- Mutli-player mode (up to 3pts)
- Customizable character appearance (up to 2pts)
- Increasing difficulty levels throughout the game (up to 2pts)
- Online multiplayer system (up to 4pts)

References

[1] Blender Foundation. 2025. *Blender*. https://www.blender.org Version 3.6.
[2] OpenJS Foundation. 2009. Node.js. https://nodejs.org JavaScript Runtime Built on Chrome's V8 JavaScript Engine.
[3] Khronos Group. 2015. glTF. https://www.khronos.org/gltf GL Transmission Format for 3D Models.
[4] jmc2obj. 2024. jMc2Obj. https://github.com/jmc2obj/j-mc-2-obj Version 126.
[5] Emscripten Contributors Kripken. 2013. Ammo.js. https://github.com/kripken/ammo.js Bullet Physics to JavaScript.
[6] Inc. Meta Platforms. 2013. React. https://react.dev JavaScript Library for Building User Interfaces.
[7] Mojang Studios. 2011. Minecraft. https://www.minecraft.net Version 1.20.1.
[8] Ricardo Cabello. 2010. Three.js. https://threejs.org JavaScript 3D Library.
[9] Sketchfab, Inc. 2025. Sketchfab. https://www.sketchfab.com Online 3D model platform.
[10] Evan You and Vite Contributors. 2020. Vite. https://vitejs.dev Next Generation Frontend Tooling.