Teru’s Journey Written Report

Ian Nicolae, Ambri Ma, Nicole Klausner

COS 426, Fall 2022

I. ABSTRACT

Teru’s Journey is a multiplayer immersive video game that is primarily written in TypeScript as a web application and is graphically represented using WebGPU features. It is primarily inspired by the 2D components of *Cave Climber* as well as the stylistic and thematic elements of more modern games such as *Ori and the Blind Forest* and *Hollow Knight*. The game consists of a competition between two Terus, the talismans of good weather. In order for one Teru to defeat the other, it must race to collect five of their individually defined colored gems the fastest. Colliding with the defined obstacles slows the respective Teru doll down, making gem collection a slower process. Once one Teru defeats its opponent, it is the one responsible for ridding the world of bad weather conditions and maintaining stasis. The game, in its current form, exists solely in the 2D realm, but with more time, our next step would be to add another layer of complexity through a final 3D implementation round.

II. INTRODUCTION

During our earlier stages of game development, we had initially planned to implement a 2D platformer game with a 3D component as a culminating final scene. Due to time and resource constraints, we ultimately decided to pursue solely the 2D avenue to complete that feature completely and carefully to ensure smooth and clear gameplay. *Cave Climber* caught our attention early on as it had some aspects we wanted to emulate (including platforms, intuitive gameplay sound, and procedural generation – in our case, rain instead of snow), but also had some flaws that we wanted to address and mend in our implementation of Teru. *Cave Climber* lacked a good user control system, and it also did not have any background textures. Additionally, the game does not have multiplayer capabilities like our implementation of *Teru’s Journey* does.

Our approach primarily centered around the usage of WebGPU for graphic development as well as WebGPU Shading Language (WGSL) language for shaders. WebGPU was a challenge in and of itself as it is an API that is still currently in development. Additionally, we utilized TypeScript for a majority of our code. The game’s backend works on NodeJS with WebSockets (in particular, the WS library). As for the elements pertaining to the physics of the game, that logic was implemented on the server side and is currently on the main thread. We had planned to put each game instance on a different worker thread, but had ultimately decided that maintaining it on the main thread made most sense for implementation.

As for the surfaces and colliders present, we had originally attempted to use Bezier curves (as there is support in the code for adding particular control points for them), but ultimately decided to pursue straight line colliders (or first degree Bezier curves) to implement our final racing game concept. This approach to the colliders performs well for the intentions and expectations of our final decided game.

III. ADVANCED FEATURES COVERED

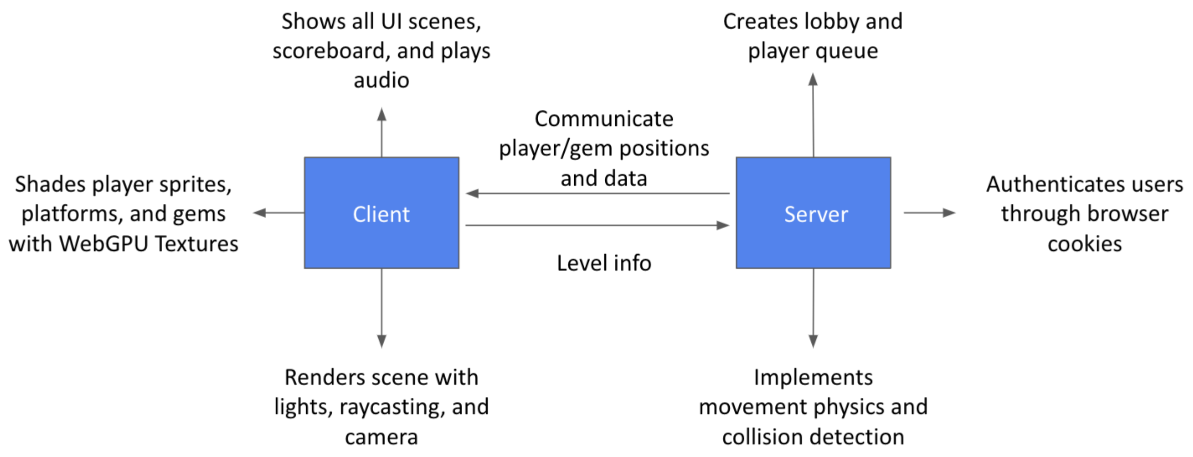
Per the assignment requirements and specifications, we completed N >= 6 of the advanced features, including:

1. Networked Multiplayer Capability – The server is hosted on Ian's computer and stress-tested to accommodate up to 100 game lobbies simultaneously.
2. Collision Detection – The character collides with objects in the playable foreground: the stone platforms, vertical walls, ceilings, and gems.
3. Vertex and Fragment Shaders – As detailed in the Methodology, we utilized GLSL to shade the character, background, and gems.
4. Advanced Image Techniques – Within the WGSL shader, we implemented parallax background textures within our gameplay found through external research, which effectively, from a visual standpoint, serves as a billboarding feature.
5. Sound – The game features relaxing background music that loops until gameplay is discontinued.
6. Onscreen control panel – At the top right corner of each player's screen, the scoreboard displays each player's score from 0 to 5 (gems collected) to keep track of game progress, as the game is designed for multiplayer competition.
7. Procedural and physically-based modeling – The 10 gems are populated into the environment using a procedural random generator based on the scene itself, as the gems must be in reachable spots. This adds a layer of depth to the game because the gems are placed differently each time you play, allowing you to explore the environment thoroughly.

.

IV. METHODOLOGY

As described above, we implemented our project backend completely in TypeScript and it runs on Node inside Docker containers. Here is a diagram explaining the basic communications of our systems architecture:


Shows all UI scenes, scoreboard, and plays audio
Creates lobby and player queue
Communicate player/gem positions and data
Shades player sprites, platforms, and gems with WebGPU Textures
Client
Server
Authenticates users through browser cookies
Level info
Renders scene with lights, raycasting, and camera
Implements movement physics and collision detection

Simplified System Architecture and Communications Between Client and Server

There were several pieces that lent homage to the project's and gameplay's success:

1. *User Control*

The user controls are kept simple and straightforward: players can move left and right, as well as jump up using the arrow keys on their keyboard or using the WASD keys (both are accounted for in the event handler). On each individual key-press event, we determine which key was pressed to complete the desired movement in the corresponding direction. We implemented the control movements in a way for the player to constantly be followed by the camera: namely, if the user moves left or right, the camera moves along the plane to match that of the character.

2. *Backend Development*

The mode in which the backend was developed was especially crucial to allow for multiplayer capability. Utilizing WebSockets allowed for the bilateral synchronous communication pipe between the client and server. This aspect of the implementation was especially important in order for the game to have multiplayer functionality. We chose to work with WebSockets as opposed to just HTTP since it allowed for more flexibility for multiplayer games happening in real time. When working in WebSockets, the client is sending packets of information to the server while the server is simultaneously broadcasting that data. This is not the case in HTTP, where the general protocol involves the client first sending a request to a server to receive data (which could work in games that require turn-taking like chess, but not in our case).

3. *Lighting*

Within the game, in order to see characters and the scene, we implemented an array of lights and casted rays horizontally coming from the left of the screen. This, in addition to the shaders explained below, laid out the game onto the HTML canvas.

4. *WGSL Shaders & Character Creation/Movement*

When conducting research on which shader language would fit the needs of our game most, we decided to ultimately pursue a project that primarily focused on WGSL shaders. While WGSL and GLSL are very similar in practice, they do have some syntactical differences. Within the WGSL shader technology software, we implemented parallax background textures within our gameplay found through external research, which effectively, from a visual standpoint, serves as a billboarding feature. Additionally, we brought our Teru character into the scene with the shaders. The Teru character was developed as a series of sprite sheets via the Procreate application to account for movement in all possible directions it could take. There were several different variations drawn up of Teru to allow for multiplayer differentiation in gameplay. The Teru doll's movement and animation was all conducted in WGSL.

Teru Sprite Sheet

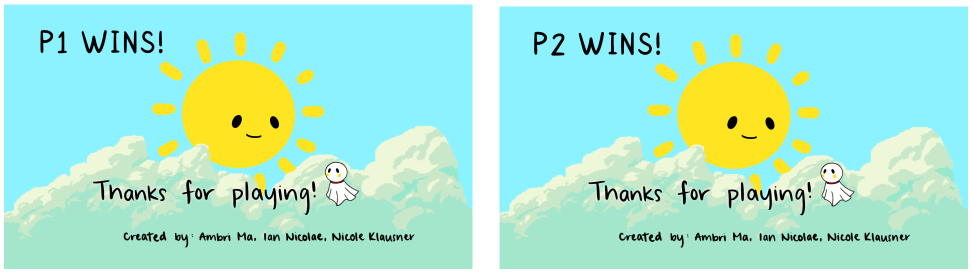
Game Gems

5. *Physics Coding and Logic*

The physics component of the gameplay posed another challenge. The most important aspects of the game where physics came into play were colliders as well as character movement. Our original debate when settling on the Teru character was how to best simulate its movement considering it had no legs and we wanted to give it a "floating" motion during gameplay. By manipulating the sprite sheets and physics logic, we were able to get a nice result which fit the storyline of the game itself. Colliders were also a challenge during portions of implementation because for a large part of the time, we attempted to unify Bezier curves that made up the scene of the game with our characters' floating movement, but ultimately found that the physics component was more manageable when approaching first degree Bezier curves with the Teru character's movement pattern.

6. Scoring

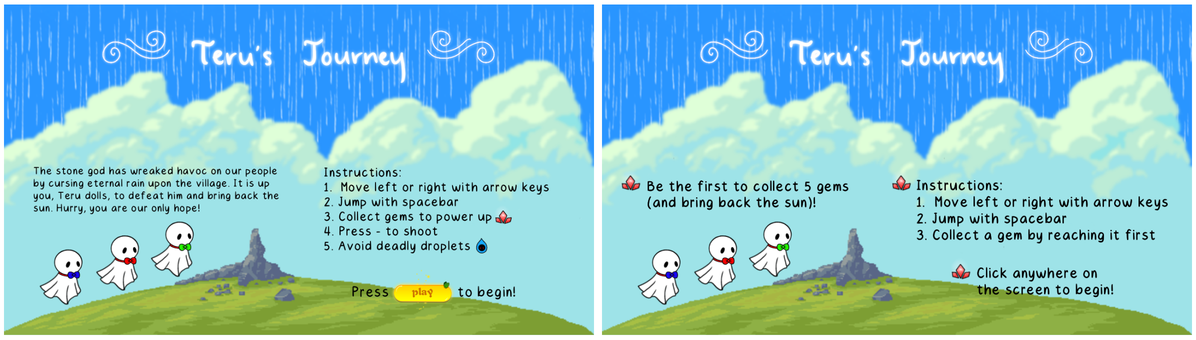
The scoring of *Teru's Journey* allows for the multiplayer game to be competitive and engaging. For the scoreboard, we added a content division (div) element on the top of the canvas where the game is drawn, and updated the HTML whenever someone scored by collecting another gem. When a player's score hits five gems, the endscreen is triggered using event handlers to return to either the P1 wins screen or the P2 wins screen, depending on which player completes the task.


P1 WINS!
Thanks for playing!
Created by: Ambri Ma, Ian Nicolae, Nicole Klausner
P2 WINS!
Thanks for playing!
Created by: Ambri Ma, Ian Nicolae, Nicole Klausner

Game Over Screen Variations

7. User Interface & Scene Creation

For the game's UI, we implemented a menu screen, a game screen, and a game over screen. Screens are toggled using mouse clicks: for example, to navigate from the menu screen to game play, one clicks anywhere on the screen to begin the game. This eventhandler also resizes the window to be fullscreen, starts the game, and starts playing sound. As for when the game concludes, the user can restart the game by pressing the "r" key on the keyboard. These commands are handled by an event handler which listens for these particular commands and sets screen booleans accordingly.



The stone god has wreaked havoc on our people by cursing eternal rain upon the village. It is up you, Teru dolls, to defeat him and bring back the sun. Hurry, you are our only hope!

Instructions:
1. Move left or right with arrow keys
2. Jump with spacebar
3. Collect gems to power up
4. Press - to shoot
5. Avoid deadly droplets

Press play to begin!

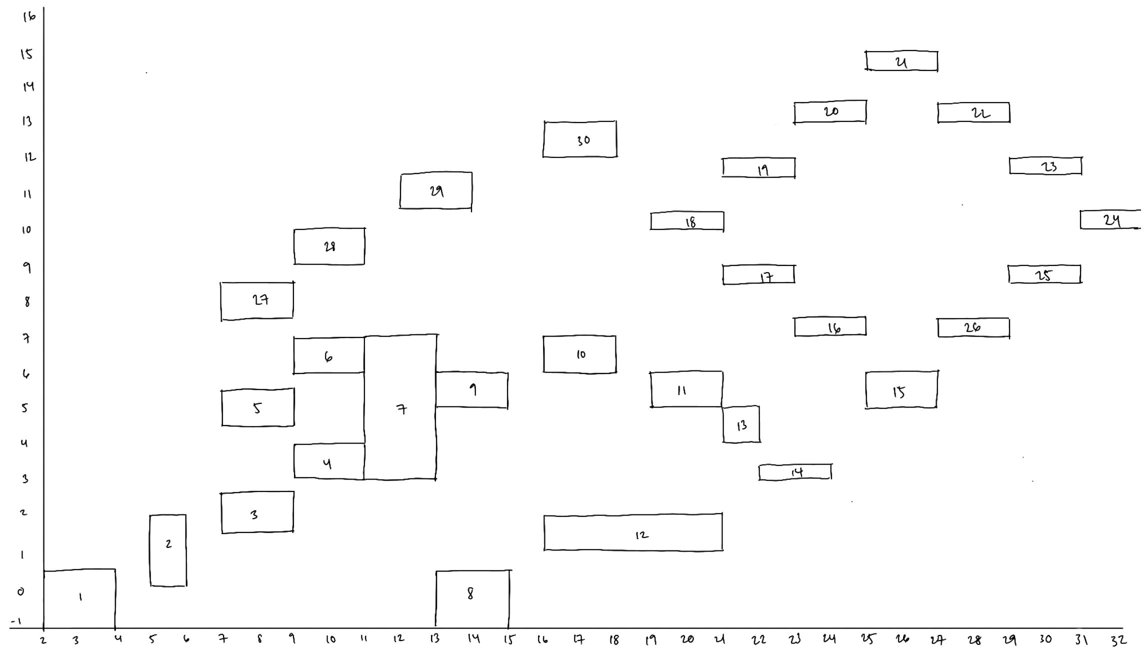
Be the first to collect 5 gems (and bring back the sun)!

Instructions:
1. Move left or right with arrow keys
2. Jump with spacebar
3. Collect a gem by reaching it first

Click anywhere on the screen to begin!

Original vs. Final Menu Screen

As for scene creation, we utilized input handles as endpoints for curves through having lines drawn on the scene. In order to account for the multiplayer functionality, we copied the scene creation for the backend and frontend development. Additionally, the coordinates needed to be accounted for counterclockwise for creation. The creation of the scene also required constantly checking if the character collides with any lines drawn with the handles.



Scene Design: input coordinates for each box into levels.ts on both client- and server-sides

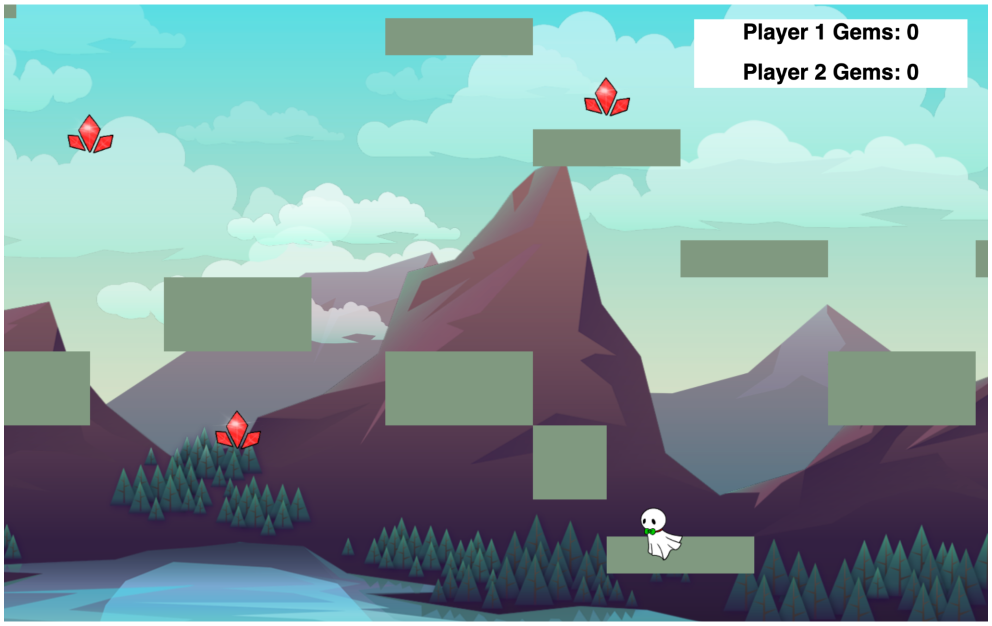
8. *Game Sound*

Game sound was an important feature that we wanted to implement to enhance the user playing experience. In order to do this in TypeScript, we imported .mp3 and .wav files into the application via a webpack, which processes file types by converting them into modules for the web browser. It essentially allows for the active transfer of messages without HTTP long-polling. In order to maintain the sound for the duration of the game, we looped the audio.

We also added sounds when a gem was collected or the game ended on top of the main background audio.

V. RESULTS

Due to server hosting difficulties and incompatibility with WebGPU, the online demo may not work for some devices. If this is the case, please see the demo video Ambri made at this link (may require download): COS426 Final Project Demo.mp4 or this Loom link: https://www.loom.com/share/dc47583074004fc78248b56b9eee8d55 . Note that gameplay in the first parts of the video lags significantly due to screen recording, but the movement controls are quite smooth in real-time play, seen at the end. Also note that the demo was created before we were able to add gem textures, so they do not appear in the game. To account for this, here is a screenshot of the game with the gem textures added.



Scene with background, character, and gem textures all present

Overall, we are very pleased with the results of *Teru's Journey*, despite it not directly matching what we had originally intended. We have produced a fully functional multiplayer game with advanced user interface capabilities. We have succeeded in developing a platform that allows for multiplayer compatibility, and also how to bring the scene to life using parallax background textures as well as sound/sound effects throughout the game.

We primarily measured success qualitatively throughout the project's development. All of our ideas for the game were developed via a series of conversations about what method would be most beneficial to complete a certain task in. When we would take a certain route for implementation, we would see if what we produced was what we had intended, and if we thought there was more development and refinement to be done, we would adjust our framework and explore other avenues (for example, our earlier mention of Bezier curves was a challenge we overcame through our development process). We also relied on feedback from our peers who are not enrolled in the course to hear their commentary pertaining to the actual graphics of the game, which was essential throughout game development. One primary recurring comment we received at the beginning was that the game would benefit from an upbeat or mysterious sound, so we took that into consideration for our final implementation of the game.

VI. DISCUSSION & CONCLUSION

Overall, we believe that the approach we took to implement *Teru's Journey* is very promising and has proven to be successful. We are also very pleased with the aesthetic and graphic components of the game. The mode in which we implemented gives a lot of control over everything in the software stack, thus allowing for the game to be optimized and improved upon in future reinterpretations and reworkings. There is a lot of room for improvement and optimization within our code, and we recognize that each new step of optimization takes time, but we definitely see potential for growth following the approach we took for implementation of a multiplayer game. One avenue that could potentially be explored is that of the dynamic reconstruction of shaders, that way the shader could be constructed depending on the particular assets needed.

Some of the next steps for the improvement of the gameplay and graphics of *Teru's Journey* include support for second and third degree Bezier curves as colliders. Additionally, implementing a scene in 3D for Teru's journey is also promising and the code undoubtedly has the bandwidth to handle such an implementation. Also, creating a backend to account for some sort of leaderboard could allow for more competitiveness across the server in *Teru's Journey*. Ideally, we also want to revisit the mode in which objects are drawn in the scenes as we think it can be optimized: in particular, we would want to implement a check to see if a pixel is inside an object by computing real world coordinates for it and shooting a horizontal ray and intersecting it with the first collider (and with that information, figure out if it it from the inside or outside by computing the cross product of the ray direction with the tangent to the collider at that point). Lastly, we had planned to implement damage tokens to complicate the game a bit more as well as feature a health bar, but we ultimately prioritized having gems present in the game as well as a scoreboard.

We have learned a lot throughout the completion of this project. For our entire group, it was our first introduction to or interaction with working on a backend with NodeJS and WebSockets, so there was definitely a learning curve at the onset of the project's development. Additionally, we learned quite a bit about the WebGPU API as well as WGSL, and how to manipulate and utilize those tools to ensure satisfying character movement and jumping inspired by *Ori and the Blind Forest* in particular. We also gained a lot of knowledge of and experience with parallax structures, a topic we did not directly ever discuss in the course material. The project allowed us to build on the computer graphics knowledge we gained over the semester and expand on that knowledge via a creative medium where we chose what aspects of the course interested us most (and we were able to pursue those interests).

While we did not completely meet our original goals, we are nonetheless happy with our results. We learned a lot throughout the process and put in a lot of effort, and it paid off graphically, procedurally, and visually. We want to thank the entire COS 426 staff, and especially Professor Rusinkiewicz for all that he has taught us this semester. The feedback and advice we received throughout the development stages of this project were invaluable, and allowed us to produce something we are genuinely happy with.

VII. CONTRIBUTIONS

Ian set the foundation for the project's backend, implemented the physics of jumping and gliding, resolved collision detection with surfaces (particularly to support bezier curves in the original setup), and mapped textures provided by Nicole and Ambri.

Ambri was responsible for designing the playable characters, main/ending menus, playable items (including gems and droplets), and game scenes. She also implemented sound, scoreboard updates, client and server collision with gems, and the scene (placing gems randomly & drawing platforms) itself through coordinate mapping.

Nicole provided support pertaining to sourcing textures, backgrounds, and special effects (including sound and projectile animations in the original setup) of the game, and she also oversaw composition of the presentation slides and writeup, with input from all group members.

VIII. WORKS CITED

A. https://dev.to/sauravmh/building-a-multiplayer-game-using-websockets-1n63
B. https://escholarship.org/uc/item/5311b662
C. https://free-game-assets.itch.io/free-horizontal-game-backgrounds
D. https://symbl.ai/blog/how-to-stream-audio-browser-with-websockets/
E. https://soultyragevin.itch.io/sukuriburu-font

F. http://freemusicarchive.org/music/Podington_Bear/Woodwinds/FluteFleet
G. https://github.com/Derndeff/cave-climber
H. https://nodejs.dev/en/learn/