“Shootout” - a 2D Shooter

By Neo Gullberg and Johan Gustafsson

2024-02-26

Objective and Requirements.

The point of our project was to program a game where you as a player have a goal to battle against an enemy in a shootout by trying to shoot them down while also trying to avoid their oncoming attacks by moving around vertically on the screen. The main *must* requirements for the game were the following:

- The user must be able to control the game with the buttons on the ChipKIT board
- The game must support multi-player mode, a scoreboard, and health bars
- Single player mode against a bot with different difficulty levels

Optional features that *may* be included, if time allows:

- Power up abilities
- Player model customization

Solution.

Our game was designed to run on the ChipKIT Uno32 platform with input and output handled by the ChipKIT Basic I/O Shield. The OLED screen of the Basic I/O Shield was used for displaying the graphics.

The most important part of our game is the gameplay of course, which consists of two players battling it out with guns and jetpacks against one another to see who dies and who lives, the latter being the winner. This part of the game mainly consists of handling the player physics and collision detection between players and bullets. The player physics routine does three main things: let gravity affect the player, let the player either use their jetpack or not, and restrict the player to the playable area. The bullet logic routine takes care of updating each bullet that a corresponding player has shot, until it inevitably gets destroyed–either by colliding with the opposing player or going off the screen.

I/O was very easy for us to implement since we had so much experience doing it from the lab sessions, however we decided to encompass most of the stuff corresponding to that in its own C struct, which we found very useful. This led to us doing the same in many other parts of our code since we found that it helped with the structure of our codebase.

Single player was implemented entirely without randomness, meaning it is entirely dependent on the state of the game, more precisely the state of the player being controlled by the buttons on the ChipKIT. Even though there is no dependence on randomness in the game, it still feels like there is since it’s near impossible for the person playing to precisely emulate the same inputs as previously that would lead to the same outcome.

Verification.

Rigorous playtesting, based on our experience we adjusted the values corresponding to the things we tested e.g., player speed, hitboxes, and the bullet’s size and speed. Furthermore, we added a debug mode. The first use case for the debug mode was to display the amount of bullets in the game world, which we used to confirm that bullets got destroyed correctly after either going off the screen or hitting an opponent. Another use case was to display each player hitbox to verify that collisions were working as intended. When it came to how the hitboxes were displayed, we had to get creative since we only had two colours to choose from, and what we ended up doing was too invert the space on the screen that the players took up, because we didn't have the “screen real estate” to display a bounding rectangle around each player.

Contributions.

Johan decided to focus on implementing collision between players and bullets, which ended up being with the Axis Aligned Bounding Box (AABB) algorithm. The reason why we used AABB was because our hitboxes could be represented by rectangles that were axis aligned relative to the coordinate system of the screen, and since the algorithm was trivial to implement in code. The game's player physics and game feel went hand in hand since player physics is a big part of our game and as he implemented and tested the physics of the players, the game feel evolved in accordance with that.

Neo wrote functions that made it easier to display pixels to the screen and implemented a sort of image format (which we ended up calling bitmaps). He also wrote stuff that made it easy to read and write input and output on the ChipKIT, and worked on a menu system. He also implemented a bunch of “helper” functions that were used to ease the coding process, one example being a utility function that was used to align different elements in the game world, which took some inspiration from Cascading Style Sheets (CSS). Lastly, he also implemented a single player mode where it’s possible to choose and play against two different AI difficulties being controlled by the computer.

Together we also worked on the overlying code structure and created a settings system to modify different aspects of the game without explicitly changing stuff in the code and having to recompile each time, which almost works as different game modes in a sense.

Reflections.

The game was very fun to develop, in large part due to us being able to choose what to make ourselves. We learned a lot about how to structure our C codebase with the help of good practices such as separating parts of the code into different files and connecting them together with the help of header files, which also contributed to improved clarity when reading old code. We ended up abandoning a scoreboard system choosing to replace it with a settings system where you get the ability to configure different properties of the gameplay. Another thing we didn’t implement was power ups since we didn’t come up with a good way to apply it to the existing gameplay.