Voxel Arena

~~(SUPER MEGA ULTRA DEATH ROBOT KILLING ZONE PLATINUM EDITION 2.X)~~

By: Paul Kasum, Aaron Piotrowski, Vernon Wong, Jacky Chiu

High Concept

A voxel-based, two-player (possibly four-player), three dimensional battle arena with destructible/constructible surfaces. (Minecraft inspired terrain generation with destructive properties; Towerfall/first person shooter/brawler battle mechanics.)

Genre / Tech Goals

Action / Shooter (First Person Style)

Multiplayer

Xbox Controller usage

Procedural Generation of Terrain

-Ability to destroy and construct terrain in game

New Content:

- Main Menu/Settings Screen (disjoined currently for tweaking)
- Control balancing (Slowed down movement and aiming)
- Power Ups spawn in terrain
- Visual change in terrain/skybox/lighting (green focus)
- Bottom boarding floor indicator
- Some robot body parts culled from player’s camera
- New GUI setup implemented, to deal with GUI bugs in executables
- Models for each of the 3 melee powers (Sledgefists, Macefists)
- Music and sound effects implemented

Aaron’s Contributions/Challenges

- Health and ammo GUI
- Player camera setup
- Character model animations and sound effects
- Minimap
- Aesthetics (Skybox, fog, music)
- Melee mechanics/graphics
    - Destroying block at certain angles
    - Cooldown on punching terrain vs punching other player
    - Melee weapon prefabs

Challenge I encountered:

GUI icons worked fine in Unity editor, but in executables did not refresh properly.

How I dealt with it:

Made another camera, put our projectile prefabs in front of it, culled everything but the prefabs (to remove terrain/skybox/players from the view), and created a loop to turn on or off the mesh renderer for each prefab according to the player’s ammunition amount, then put the camera’s output where the GUI used to be.

Paul’s Contributions/Challenges

- Basic Controls re-tweaking and balancing (Primarily movement slowdowns)
- Power Up Boxes [Spawn in Terrain, produce ammo or buff to health/fist power]
- Minor range increase to projectiles [Basic bullet breaks blocks]
- Terrain re-texturing [Green everywhere! Black blocks on the bottom two layers to warn of edge of lower map]
- Settings integration [Making sure they carry over to the actual player sequence]
- Constant hit box management on all objects
- Pause universally linking to both JavaScript and C# codes

Challenge I encountered:

The controller functionality in Unity was my biggest battle. Aside from that most of the struggle and strain merely came from iterate, test, and tweaking of values to function more smoothly.

How I dealt with it:

This was mostly a trial and error focus of re-test, re-work, and re-implement. (Half it, check it, tweak up or down from there) Playtests showed the game needed an overall slowdown in how the joystick functioned with the movements. Other than that is came from just tweaking the small bugs that would rise up.

Vernon’s Contributions

- Lava/fireblocks with damage-over-time effects
- Initial idea, and code, for falling missile functionality
- All Illustrator art assets (weapon icons, logos, buttons)
- Button.cs module that others can use to create buttons
- SettingController.cs that manages all game start settings such as health and ammo counts
- All menus:
  - Main menu
  - Settings menu (setting up starting conditions)
  - Player controller menu

Challenge I encountered:

Lava growth scenario failing / Last minute art asset demand / Missile code functionality

How I dealt with it:

Research was done regarding how to properly get lava to select blocks and replace them, along with an in depth look at how classes (for block types) are handled in .Net. Ultimately, it failed, but I know I could properly get it to work now, given time. Organizing data was a matter of trial and error to see what method of keeping track of data was optimal.

Internal Processes/Methodologies

- Copious Free Asset usage:
    - Speed up that art growth while giving more time to code dedication
- Build Storage: Google Drive
    - Allowed ease of uploading soon to merge content and storing of previous and current builds incase of ‘rewind’ scenario
- Tracking people’s workload: Facebook message board
    - This helped give immediate view of what people were doing as well as allowing us the luxury of commenting and adding feedback faster

Lessons learned about game development

- Closely monitor what others are working on, so that people don’t end up working on the same thing at the same time, or working on two things that won’t integrate well.
- Implement a feature-freeze well before the final deadline, so that you can focus on detecting and fixing small bugs, and polishing what already works.
- Use source control software like git to cut out the time it takes to merge code
- Comment your code well so that when others have to use your code, they can more quickly understand how it works.
- Playtest to detect where you should focus on re-balancing and re-working (Especially in controls)
- Like forest fires, only you can prevent feature creep. [Constantly tweak your scope]
- Check the executable’s functionality outside of your development environment with each build.
- If you are unsure what to work on, ask your group.