Project Name: Battle For The Galaxy



James Lockard  
 Courses: ComS 227, ComS 228, SE 319, ComS 104, ComS 106  
 Languages: Python, HTML, JavaScript, Java, PHP and CSS  
 Internships: None  
 Number of Semesters: 5

Ethan Ball  
 Courses: ComS 227, ComS 228, ComS 319, CprE 288, CprE 185  
 Languages: JavaScript, Java, C, HTML and PHP  
 Internships: None with actual programming.  
 Number of Semesters: 7  
  
Nick Urbain  
 Courses: ComS 127, ComS 227, ComS 228, ComS 327, ComS 311  
 Languages: Java, C++, Python and JavaScript  
 Internships: None  
 Number of Semesters: 5  
  
Pawel Darowski  
 Courses: ComS 227, ComS 228, ComS 363, CprE 288, CprE 308, CprE 381  
 Languages: C, Java, HTML and MySQL  
 Internships: None  
 Number of Semesters: 6

Description of project

A spiritual successor to or clone of Silent Death Online. “Silent Death Online was a top down arena style space shooter MMO involving players earning money by taking out other players. With this money you could upgrade your ship with many different weapons or buy a better spaceship, plus the mouse and keyboard controls kept the game fast paced and fun.”

There were many modes in the original Silent Death that we could implement:

* Deathmatch Mode: All players against one-another.
* Team Deathmatch Mode: Originally there were four factions/clans that you could join and team deathmatch allowed the clans to battle one-another.
* Construction Mode: This mode involved guarding robots that were working to build a space station. First to finish building 3 wins.
* Mining Mode: Guard your mining shuttle while it attempts mine resources from a nearby asteroid and deliver them to your space station.

Additionally, we could add a battle-royale mode and/or single-player mode if time permits. Currency was in form of credits where kills gave credits and winning multiplied credits earned. Credits were used to purchase upgrades and buy ships. To enable people to talk to one another we would implement a complex chat system with different screens for friends, faction and universe.

Language/Platform/Libraries

## Platforms

* Android
* Web
* PC

## Languages

* Java
* MySQL
* HTML

## Frameworks/Libraries

* LibGDX
* Xbox Controller

Complexity

## Gameplay

* Realtime gameplay with client-server relationship -- some balance would need to be struck between what the client handles and what the server handles so performance does not suffer.

## Client-side

* Keyboard input for moving ship forward, reverse, strafe left/right, selecting boosts/items, in-game chat and in-game menu navigation.
* Mouse input for rotating ship, shooting and using boosts/items.
* Bluetooth/USB controller input (ie XBox controller) to optionally replace keyboard/mouse gameplay.
* Server access for obtaining players’ ship/equipment/condition to draw them correctly on screen, other ship and projectile locations/velocity, items and bot/AI locations.
* Locally processing graphics.
* Screens include the welcome splash screen, options, game mode selection, pre-game lobby, loading screen, in-game screen, in-game menu and post-game screen.
  + In-game screen requires a background, a large space for gameplay, indicators for shield/armor/hull damage, weapon heat/cooldown, available items and boosts, other player names/kills and mini-map with other player locations.
* Campaign mode: time permitting, the campaign mode would be entirely client side and take over all server-side responsibilities. Action sequences would need to be added as well as dialogue boxes and linear access to other screens. For more complexity, a whole new gameplay type could be added, such as a platformer or puzzle minigames.

## Server-side

* Matching players of similar rank together.
* Client read access for updating everyone’s in-game data simultaneously with multithreading.
* Client write access for routing location/velocity of all ships, projectiles, items and bots/AI to all clients.
* Calculating when ships and projectiles/bots/AI/other ships make contact and sending correct triggers to clients (such as Ship5 is at 0 health and needs to blow up now).
* Keeping in-game data up to date on each client and spawning items on the map.
* Determining how bots/AI act based on the position of other players on the map and gameplay mode.
* Database read/write access for gathering/updating nonvolatile information to route to each player such as each player’s name, current equipment and rating.

## Database

* Tables to store each player’s nonvolatile data such as ship/shield/armor/hull purchases and available credits as well as statistics such as number of games played/won/disconnected, kills, damage dealt and achievements unlocked.
* Tables to store player ratings.

## New Technologies

* Web
  + Web controls will be similar if not identical to the desktop application.
* Android
  + Android controls will need to replace keyboard/mouse.
* Xbox Controller

## New Frameworks

* LibGDX

## New Languages

* MySQL