Project Name TBA

PBJ2 Productions

Peter Bartosch  
Ben Crist  
Josh Douglas  
Jordan Buehler

# Genre

Side-scrolling Arena Shoot-em-up

# Theme

Colorful Retro (80's) Sci-Fi

# Platform

PC (Windows)

# Game Overview

The player controls their avatar to navigate the arena while attacking enemies and avoiding their attacks. The player character is equipped with a full-body armor suit which also acts as a gun (built into the arm). Players can compete against AI bots or other humans over LAN.

# Target Audience

Twitch gamers, deathmatch enthusiasts, and lovers of the 80s.

# Game Influences

* Soldat – www.soldat.pl
* Tribes

# MuSCoW Analysis

## "Must Have" Features

* Weapon
* Bullets
* Jet boots
* Physics / Collisions
* Deathmatch mode
* Camera
  + Side view with limited perspective
* AI/Bots
* Sound
  + Effects
  + Music
* Level Saving and Loading
  + Level Editor
* Art
* Textures
* Menus
* User Interface
  + Menus
  + HUD
    - Current Score
    - Health
    - Kill Updates / Text Callouts

## "Should Have" Features

* LAN Multiplayer
* Skeletal Animation
* Multiple Weapons
  + Grenades
  + Shotgun
  + Active Weapon in HUD
* Configurable Key Bindings
* Global Lighting
* Leader board
* Weapon/Health/Powerup Pickups

## "Could Have" Features

* Gamepad Support
* Support for Online (High Latency) Play
* Player Chat & Taunts
* Multiple Skins/Textures
* Multiple Game Modes (Team DM, CTF, King of the Hill, etc)
* Secondary Fire on Weapons
* Configurable Level Parameters (gravity, etc)
* Persistent Player Stats (per-server)
  + Fame System
* Screenshots (Kill shots)
* Level Streaming (Clients download map database from server before game begins)

## "Won't Have" Features

* Storefront / Micro-transactions
* Multiple Character Models
* Story / Campaign
* RPG Elements
* Armor / Damage Reduction

# Schedule

### M0: July 15, 2013

Game concept complete  
Basic schedule set

### M1: July 29, 2013

Core engine subsystems complete

### M2: August 12, 2013

"Must Have" features complete  
Playable game

### M3: August 19, 2013

Additional features complete

### M4: August 26, 2013

Feature freeze  
Only debugging, polishing, and content/level design from here on

### M4: September 2, 2013

Release!

# Team Responsibilities

### Peter

Networking Subsystem Lead  
Gameplay & Simulation  
Art/Sound/Level design

### Ben

Graphics Subsystem Lead  
Resource Loading  
Art/Sound/Level design

### Josh

Audio and Input Subsystems Lead  
GUI/Menus  
Shaders  
Art/Sound/Level Design

### Jordan

Production  
Art/Sound/Level Design  
Marketing  
PR

# Libraries & Middleware

OpenGL – Graphics  
OpenAL – Audio  
GLFW – Window Management & Input  
GLEW – OpenGL Extension Detection  
GLM – Vector/Matrix Math  
Box2D – Physics  
SQLite – Resource/Configuration Files  
STB Image – Image File Parsing  
Libsndfile – Audio File Parsing  
Open Asset Import Library (Assimp) – Level Editor 3D File Parsing  
Pugixml – Level Editor XML File Parsing

# Anticipated Technical Hurdles

* Synchronizing game state across multiple clients
  + By limiting multiplayer play to fast LAN networks and giving the server "final say" in all cases, compensating for latency should be somewhat easier.
* OpenGL 3+
  + Since most of our work thus far has involved legacy OpenGL APIs, getting used to working with VBOs, shaders, etc. will be interesting.
* Memory Fragmentation
  + With many short-lived objects—bullets, particle systems (maybe?), etc.—memory fragmentation may become a problem. We may need to implement custom allocators for certain classes.

# Engine Architecture Overview

The game consists of three primary parts, each implemented in their own executable: A server, a client, and a level editor. Additionally, there are a number of subsystems that make up the engine. The graphics subsystem handles interfacing with OpenGL, GLFW, GLEW, etc. to draw meshes on screen. The networking subsystem handles communication between server and client. The audio subsystem handles interfacing with OpenAL to create an immersive audio environment. The input subsystem involves translating keyboard and mouse input into commands to send to the server (eg. "jump," "move left" or "aim at 47 degrees." The simulation subsystem combines input from players and AI agents (bots) with the current game state and elapsed time to discretely simulate the game world.

# UML Diagrams





