

# **Shattered Seal: Echelon**

Design Document

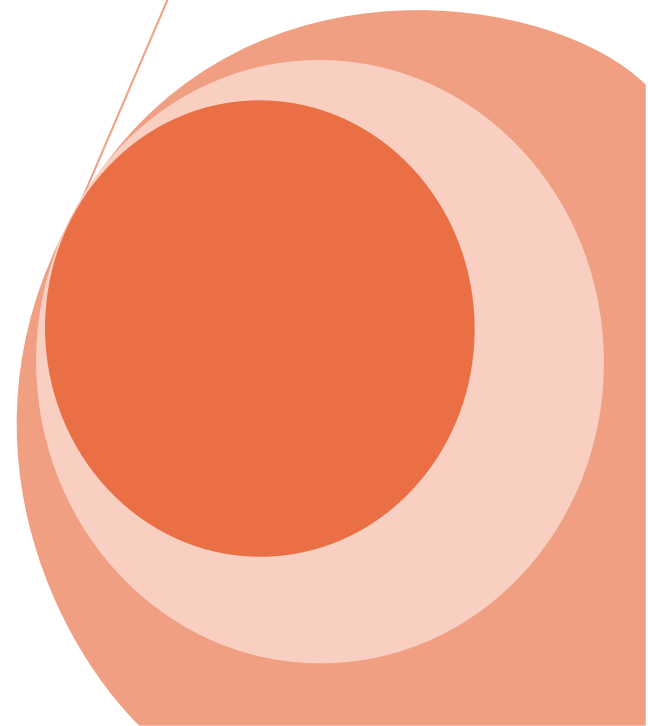
Bryan Evans, [bevans@my.fit.edu](mailto:bevans@my.fit.edu)

Dan Baumann, [dbaumann@my.fit.edu](mailto:dbaumann@my.fit.edu)

Paul Graham, [pgraham@my.fit.edu](mailto:pgraham@my.fit.edu)

Scott Leierer, [sleierer@my.fit.edu](mailto:sleierer@my.fit.edu)

Stephen Garcia, [sgarciam@my.fit.edu](mailto:sgarciam@my.fit.edu)



## Table of Contents

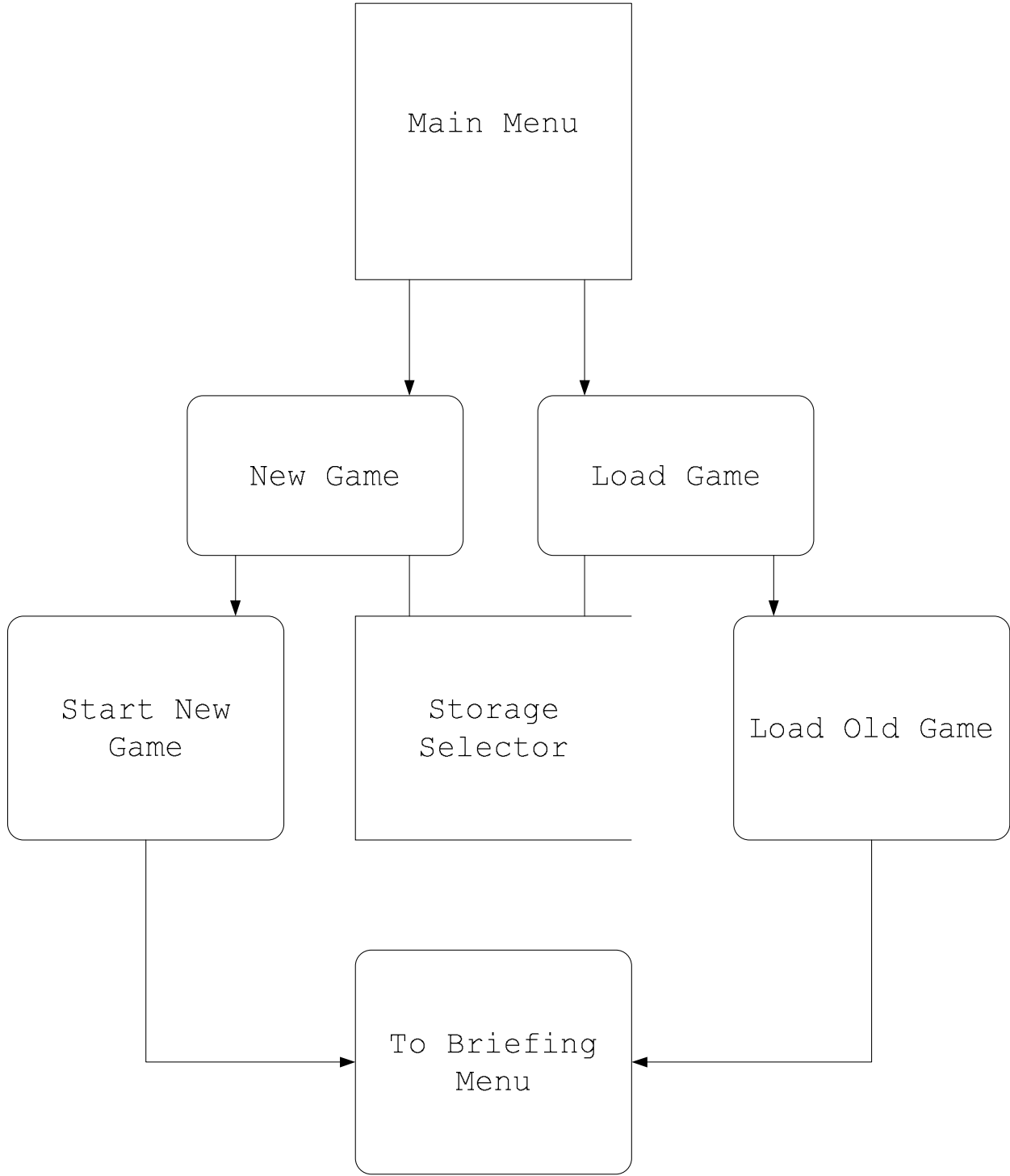
Revision History .....	3
1. Flow Charts .....	4
1.1 Main Menu Flow .....	4
1.1.1 Main Menu Flow Diagram.....	4
1.1.2 Main Menu Flow Process .....	5
1.2 Mission Briefing Flow .....	6
1.2.1 Mission Briefing Flow Diagram.....	6
1.2.2 Mission Briefing Flow Process .....	7
1.3 Mission Flow .....	8
1.3.1 Mission Flow Diagram .....	8
1.3.2 Mission Flow Process .....	9
2. UML .....	11
2.1 Character UML.....	11
2.1.1 Character UML Diagram.....	11
2.1.2 Character UML Description .....	12
2.2 Game UML.....	15
2.2.1 Game UML Diagram.....	15
2.2.2 Game UML Description .....	16



# 1. Flow Charts

## 1.1 Main Menu Flow

### 1.1.1 Main Menu Flow Diagram



### **1.1.2 Main Menu Flow Process**

**Main Menu:** The main menu offers the user a choice to either start a new game or load a previously saved game.

**New Game:** The new game option allows the user to select a storage device to save their game. Then, a save file is created on the corresponding storage device and data files are created for holding data for talent trees, level data, completed objectives, etc.

**Start New Game:** Once the new game save file has been created on the storage device, the new game begins.

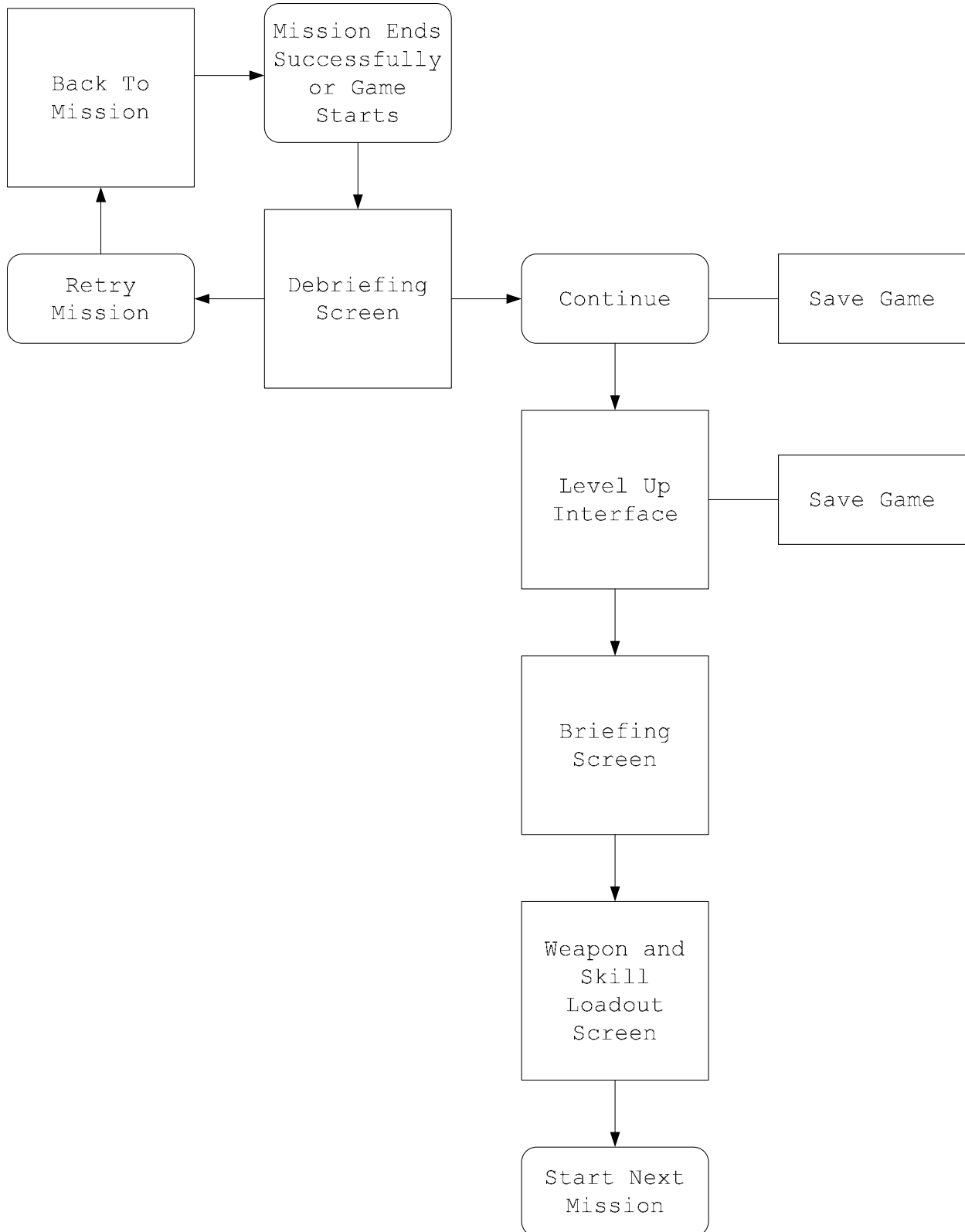
**Load Game:** The load game option allows the user to select which previously saved game they would like to load.

**Load Old Game:** Once the previously saved game that will be loaded has been selected, the corresponding files are loaded into memory.

**To Briefing Menu:** If a new game has been created, the briefing menu for the first level is loaded and displayed to the user. If an old game has been loaded, the save point is read to decide which level the user should be playing. That briefing menu is then loaded and displayed.

## 1.2 Mission Briefing Flow

### 1.2.1 Mission Briefing Flow Diagram



## 1.2.2 Mission Briefing Flow Process

**Mission Ends Successfully or Game Starts:** Once the user loads an old game or starts a new game, they are presented with the debriefing screen for the last mission they completed. If it is a new game, they are presented with the "debriefing" screen that occurs before the first mission.

**Debriefing Screen:** Addresses the end of the mission and displays complete and incomplete objectives for the previous mission. The user is given the option to either retry the previous mission or to continue to the next mission.

**Retry Mission:** When the user selects the retry mission option, the previous mission is loaded again and they may play through it again.

**Back to Mission:** The user will play the mission that they had previously completed. Once it is completed, they will be returned to the debriefing screen.

**Continue:** If the continue option is selected from the debriefing screen, the game is saved to the file and the level-up interface is shown.

**Level-Up Interface:** The level-up interface displays the user's current talent tree configuration and allows the user to spend any previously gained talent points. Once the user is done spending any talent points they wish to spend, they may continue to the briefing screen for the next mission. Once they continue to the briefing screen, the game is saved to prevent any lost talent tree data.

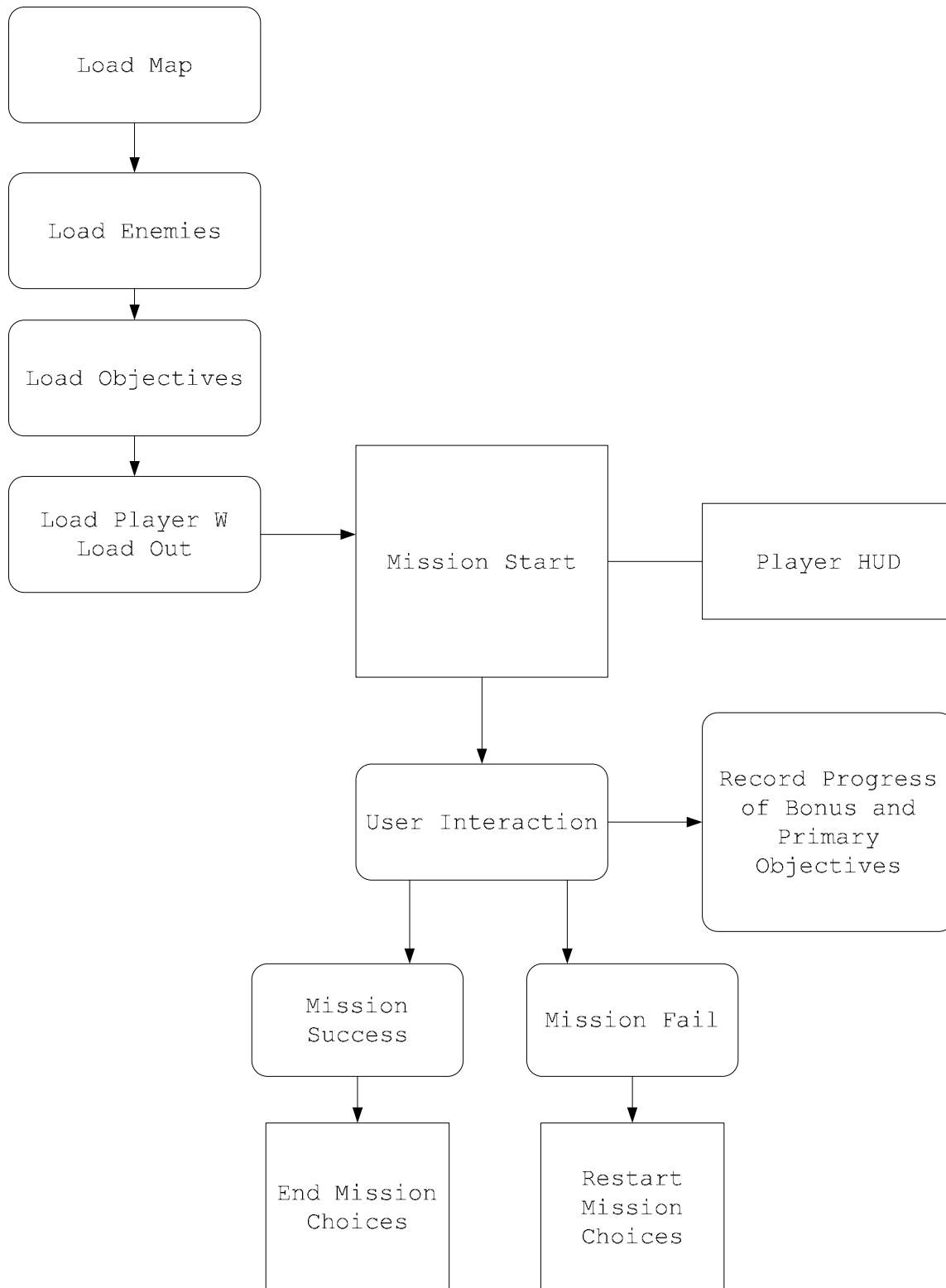
**Briefing Screen:** Displays the briefing for the next mission. Briefing includes story elements and primary and secondary objectives.

**Weapon and Skill Loadout Screen:** Allows the user to change selected loadout. Loadout includes weapon, offensive magic, defensive magic, and tech ability.

**Start Next Mission:** Once loadout is selected, the next mission begins.

## 1.3 Mission Flow

### 1.3.1 Mission Flow Diagram



### 1.3.2 Mission Flow Process

**Load Map:** Finds the correct data file for the corresponding map and draws the textures to each tile on the map. Any obstacles are then placed on top of the textures. Triggers for events or objectives are also placed.

**Load Enemies:** Enemies are loaded into memory from data file but are not drawn to the map. Enemies on the map are loaded dynamically. This step loads any enemy types into memory for easy access during the mission.

**Load Objectives:** Objectives are loaded into memory from data file.

**Load Player Loadout:** Player loadout is loaded into memory so that user controls will use the expected abilities.

**Mission Start:** The mission begins and the player character is drawn on top of the map in the center of the screen. The HUD is placed over everything. HUD includes health and mana bars.

**User Interaction:** User controls the player character throughout the level. User input will cause triggers to be activated and events will be run in response.

**Record Progress of Bonus and Primary Objectives:** As the user plays through the level, triggers will cause objectives to be marked as completed. The user may examine their progress of objectives as the level continues. When objectives are completed, the user will be notified through a sound effect and display.

**Mission Success:** When the user completes all primary objectives, the mission ends successfully and the user is notified as such.

**End Mission Choices:** When the mission ends successfully, the user is given the option to either retry the mission or continue to the next mission.

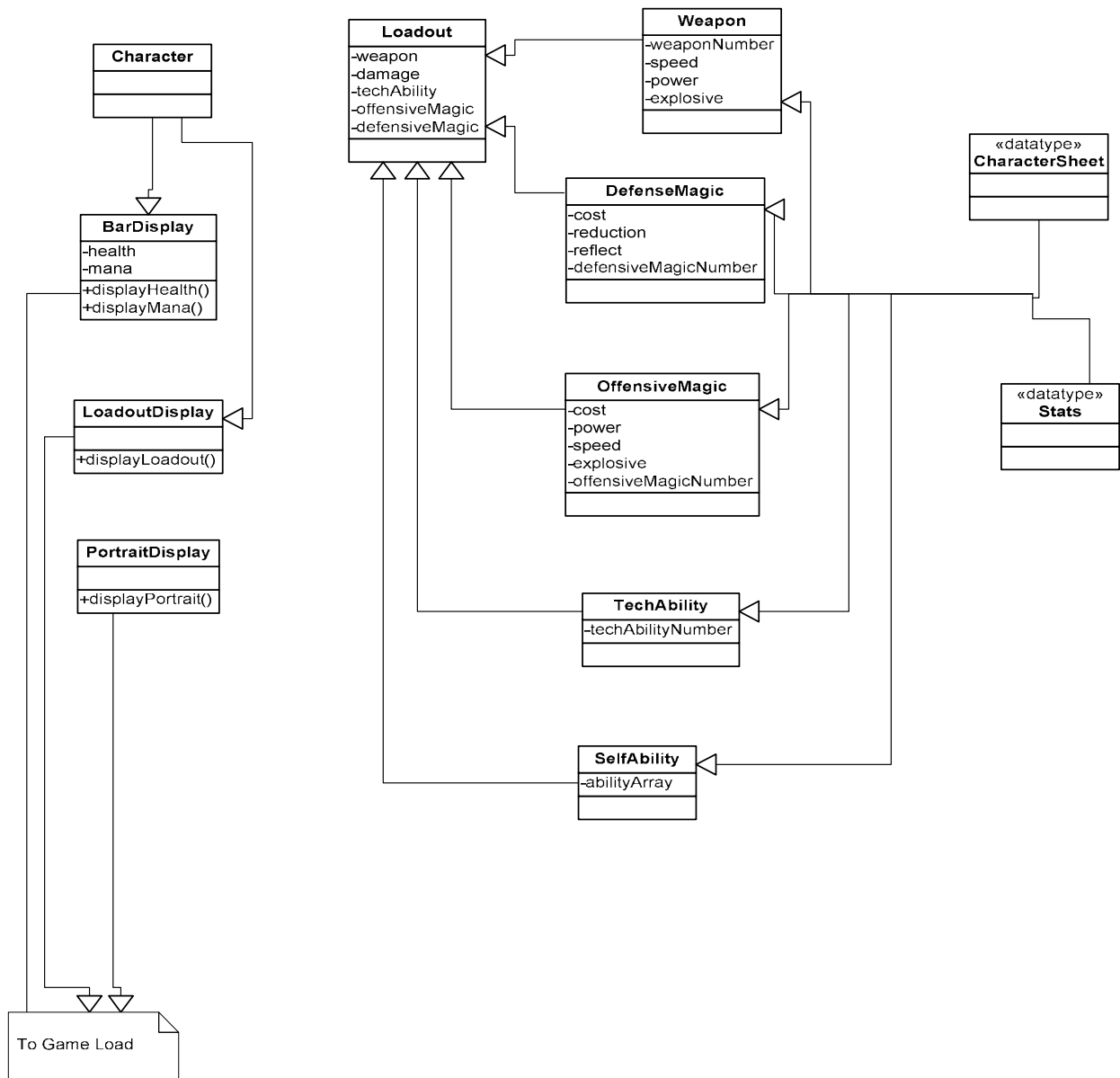
**Mission Fail:** If the user dies or fails to complete a primary objective, the mission fails and the user is notified as such.

Restart Mission Choices: When the mission fails, the user is given the option to retry the mission or to quit and return to the main menu screen.

## 2. UML

### 2.1 Character UML

#### 2.1.1 Character UML Diagram



### 2.1.2 Character UML Description

Weapon: A weapon type

int weaponNumber: Identifies which weapon the object represents

int speed: The speed that the projectile moves

int power: The damage the projectile does

Boolean explosive: Whether the projectile has an explosive radius or not

DefenseMagic: A defensive magic type

int defensiveMagicNumber: Identifies the defensive magic

int cost: The mana cost of the spell

int reduction: Amount of damage reduced when spell is active

Boolean reflect: Whether the spell reflects projectiles or not

OffensiveMagic: An offensive magic type

int offensiveMagicNumber: Identifies the offensive magic

int cost: The mana cost of the spell

int power: Damage done by the spell

int speed: Speed of the spell

Boolean explosive: Whether the spell has an explosive radius or not

TechAbility: A tech ability

int techAbilityNumber: Identifies the tech ability

SelfAbility: All of the self abilities that the player character has

int[] abilityArray: An array of all the talent points that have been used in self abilities

Loadout: The current selection of weapon, magics, and ability that the user has chosen.

Weapon weapon: The current weapon selected

DefenseMagic defensiveMagic: The current defensive magic selected

OffensiveMagic offensiveMagic: The current offensive magic selected

TechAbility techAbility: The current tech ability selected

double damage: The damage modifier modified by other abilities

Character: Holds other classes related directly to the player character

BarDisplay: Used to display and keep record of the health and mana of the player character

int health: Amount of health the player character has remaining

int mana: Amount of mana the player character has remaining

void displayHealth(): Displays remaining health on the health bar

void displayMana(): Displays remaining mana on the mana bar

LoadoutDisplay: Displays the current loadout to the user

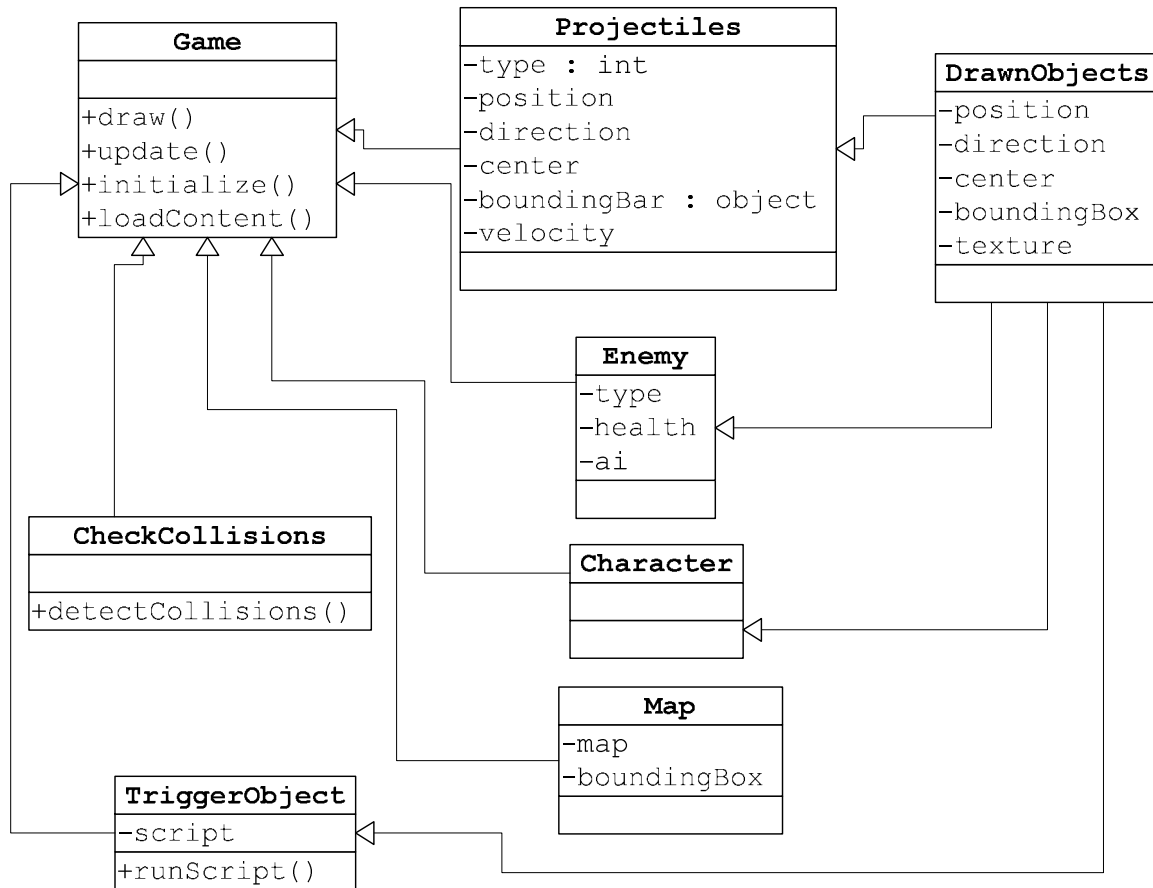
void displayLoadout(): Brings up a display over the game, pauses the game, and displays the current loadout

PortraitDisplay: Displays the character portrait next to the health and mana bars

void displayPortrait(): Loads the user's portrait and displays it

## 2.2 Game UML

### 2.2.1 Game UML Diagram



## 2.2.2 Game UML Description

DrawnObjects: A super class for any object that needs to be drawn onto the map.

Vector2 position: The object's position on the map

Vector2 direction: The direction the object is facing

Vector2 center: The center of the object

Shape boundingBox: The boundary box of the object, used for collision detection

Texture texture: The texture designated to the object

Projectiles: Used for different types of projectiles that are on the map

int type: The designating number for the projectile type

double velocity: Speed of the projectile

Enemy: Used for different enemies on the map

int type: The designating number for the enemy type

int health: Remaining health the enemy has

int ai: The designating number for the AI the enemy uses

Map: The map for the level that is being run

Texture map: The texture used for the map. Created through combining many smaller tile textures

Shape[] boundingBox: Boundary areas of the map that the user cannot enter

CheckCollisions: Detects collisions between objects

void detectCollisions(): Checks whether two objects on the screen have collided. Based on which two objects have collided, action will be taken.

TriggerObject: Used to identify a part of the map that triggers an event or completes an objective

XMLFile script: File that holds the script for what to do for a specific trigger

void runScript(): Called when the trigger is activated, completes the triggers script