SOFTWARE ARCHITECTURE DOCUMENT

SIX GUYS ONE ASTEROID

08

**Fall**

Jeffrey Leung [ID 260402139]

Andrew Dennis [ID 260475864]

Corey Perlman [ID 260475188]

Neil Sabharwal [ID 260474982]

HoaiPhuoc Truong [ID 260526454]

Richard Pringle [ID 260302515]

Table of Contents

[System Architecture 3](#_Toc349930429)

[System overview 3](#_Toc349930430)

[Views 4](#_Toc349930431)

[Software Subsystems/modules 5](#_Toc349930432)

[Element subsystem 5](#_Toc349930433)

[Multiplayer Subsystem 6](#_Toc349930434)

[Game Score Subsystem 7](#_Toc349930435)

[Graphics Subsystem 7](#_Toc349930436)

[Physics Subsystem 8](#_Toc349930437)

[UI Subsystem 9](#_Toc349930438)

[Analysis 10](#_Toc349930439)

[Design rationale 11](#_Toc349930440)

[Workload breakdown: 12](#_Toc349930441)

# System Architecture

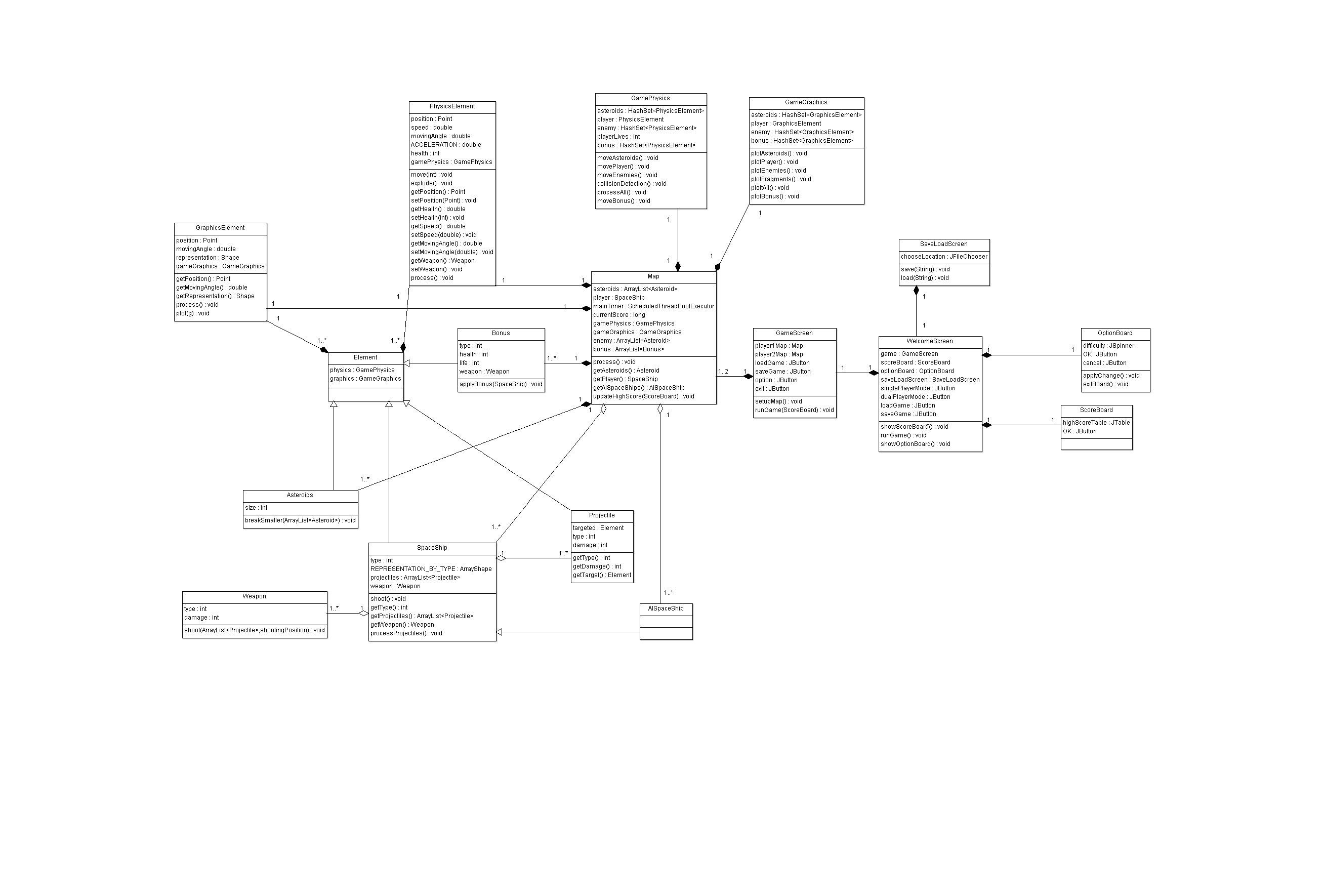
## System overview

This project entails making a computer game called Asteroids through the use of the java programming language. In the game the user must have the ability to control a spaceship and must be able to shoot asteroids.

The system will be using Java’s Swing components for rendering visual data on the screen. The system will have one class for managing visible objects on the screen. These visual objects will have positions and specific visual properties that define them, and specific physical properties/behaviors. These visual objects include the spaceship, the asteroids, aliens, bullets, etc.

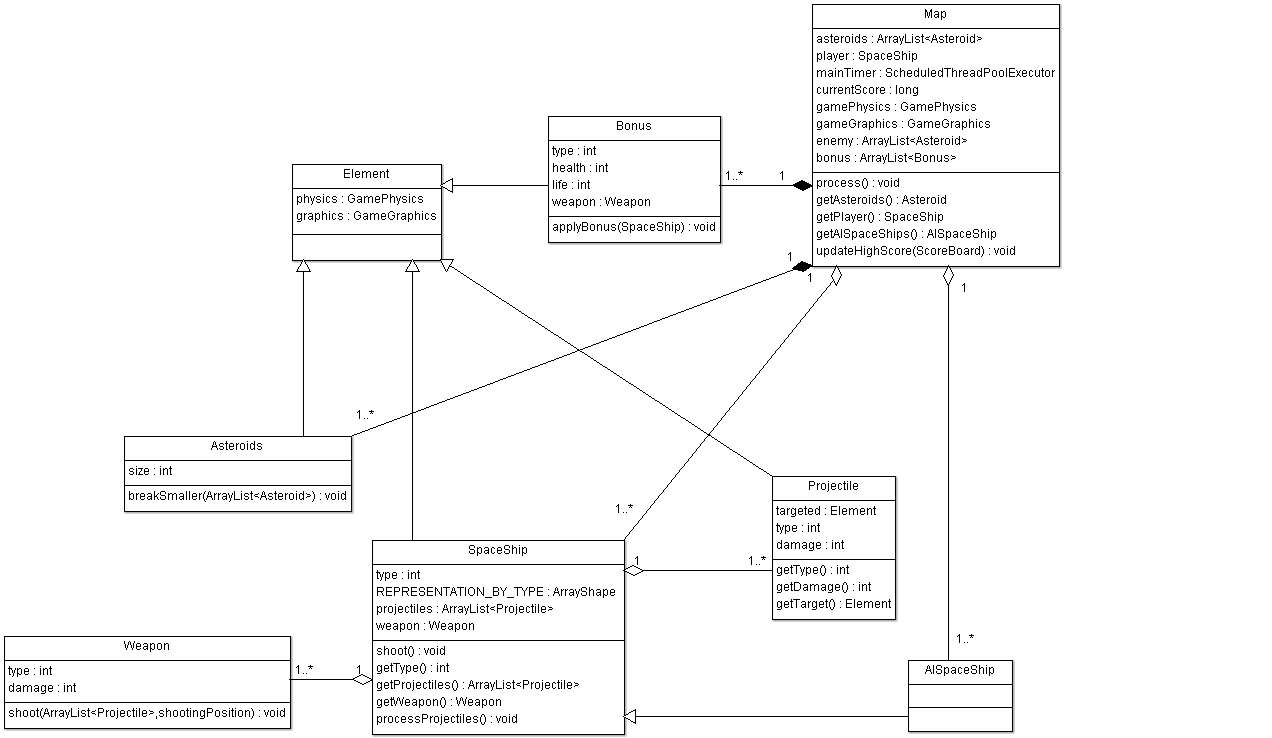
We can separate the game into two main parts for the user. The first is the UI where there will be several menus where the user can choose his options, what mode he wants to play, and the list of high scores. The second more important part is the actual game. For this we have the main “map” which will be the black background where all visuals will take place. This will have all the objects on it which we have classified as “elements”. This contains all objects such as the spaceship, aliens, asteroids etc. There will then be a physics subsystem which will take care of the behavior of these elements and the displaying of said elements. There is also a game physics which determines the overall interaction between elements and their separate physical properties.

## Views



# Software Subsystems/modules

## Element subsystem



The diagram above shows the element subsystem. This subsystem contains SpaceShip and Asteroid related classes. Most of the classes (Bonus, Asteroids, SpaceShip, Projectile) within this subsystem will be inherited from the Element class which will called methods within the GamePhysics and the GameGraphics classes.

All elements in the game are defined as general objects. These elements include but are not limited to the player’s ship, asteroids, enemies, and bullets fired at the asteroids. Subclasses for the different types of objects will be created. Objects can be set to active or inactive, created or deleted at any time. Objects have a position element on the screen. Positions are calculated independently for each object based on their type and what they react to.

The Bonus subsystem will use the decorator method to implement all different items for different purpose such as increase in health or speed or defense or all at the same time.

The Projectile subsystem is responsible for keeping track damage that applying to the spaceship and what the spaceship is aiming at.

The Weapon class will contain information about the weapon equipped by the spaceship: shooting patterns, projectile type and special effects (if applicable).

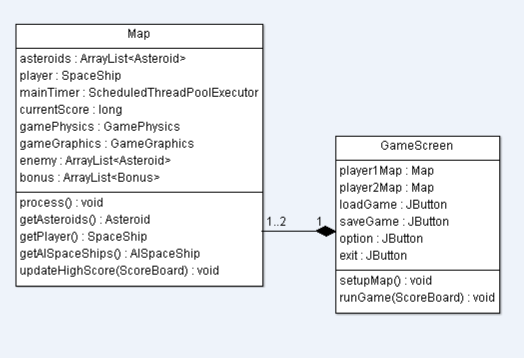
The SpaceShip class will contain methods that know what weapon and where the spaceship is shooting at the moment. Since this is a class that is inherited from the element, GameElement can easily have to information of the spaceship and do other related calculations on it.

The asteroid class is similar to the spaceship class because both are inherited from the element class.

## Multiplayer Subsystem

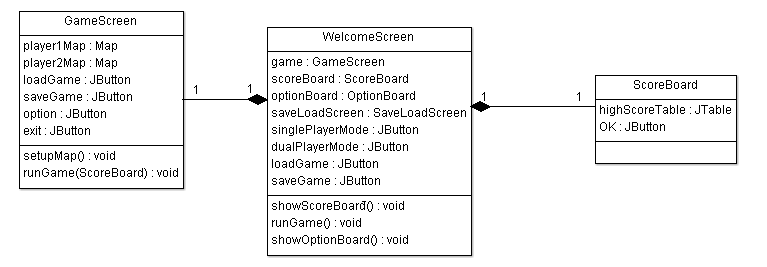
The purpose of the multiplayer subsystem is to create an alternating mode where players compete against each other’s high scores. Each player will have the chance to play through the game until they lose. Their score will then be saved and compared to that of their opponent.

This subsystem simply creates the option for two people to play against each other. The multiplayer will thus simply set up 2 maps (the second one will be made after the first one is finished) and will then compare the two scores and announce the winner.



## Game Score Subsystem

The game data subsystem will keep track of the score data of the game. It will also store previous scores to be used in a high score view. Additionally it will keep track of unlockable items achieved through a certain score level.



## Graphics Subsystem

The graphics subsystem is responsible for displaying and organizing option buttons and the general display of the game. These objects will be rendered visually using java swing components. The element class will be combining methods from the GraphicElement and PhysicElement and communicate with the object system (spaceship, asteroid etc.). The GraphicsElement will communicate with the element class and get the position, moving angles from it and also the data that have been processed by the PhysicElement through Element class.

The GameGrpahic class will display all out customised spaceship, asteroids tec. on the map and those plot/graphic will be moved by both the GraphicElement class and PhysicElement class.



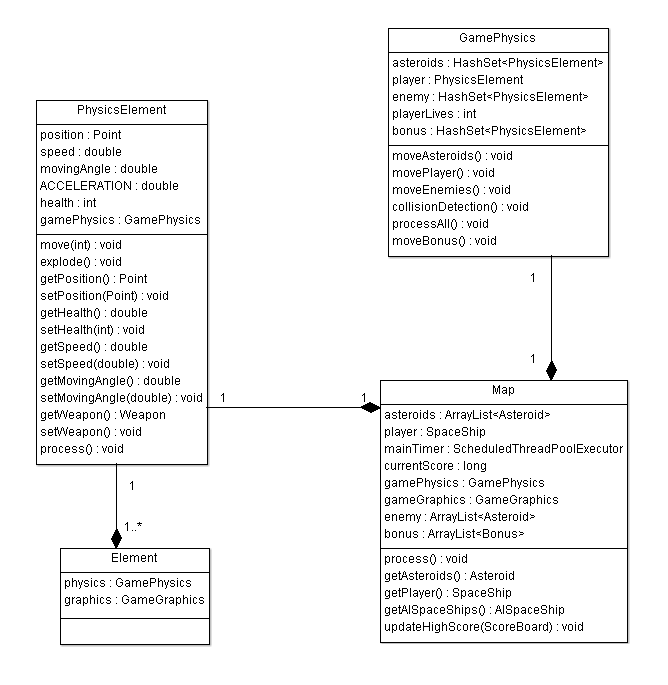
## Physics Subsystem

This subsystem will be the control class for all moving objects. We will be using it to have the object behave / move according to the laws of physics (momentum, directionality, etc). This means to slow something down or change directions there must be an acceleration or force applied and the physics subsystem will take care of establishing the outcome of these interactions.

Physics Element is used to describe the physical properties for individual elements. This will track the positioning of the element and influence how it acts in the physical world of the game.

Game Physics will calculate and determine the interactions of multiple elements and the outcome of their physical behavior after interactions. This means if a bullet collides with an asteroid it will split of be destroyed, etc.

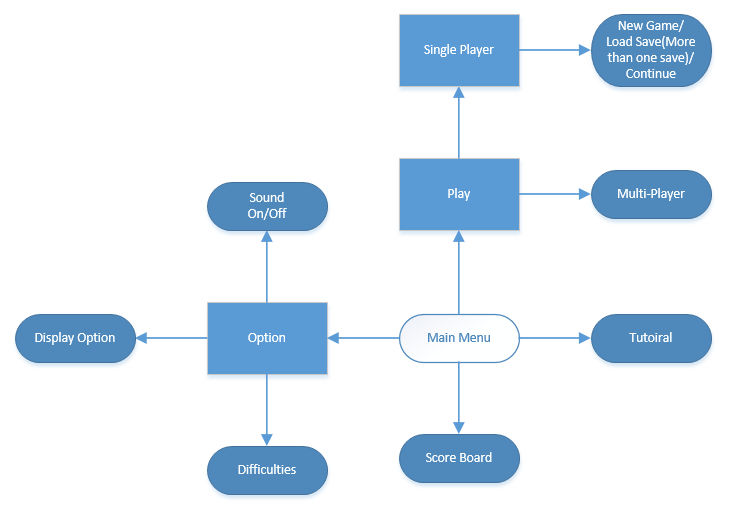
The diagram below shows the physics subsystem and its interaction with the map and element classes



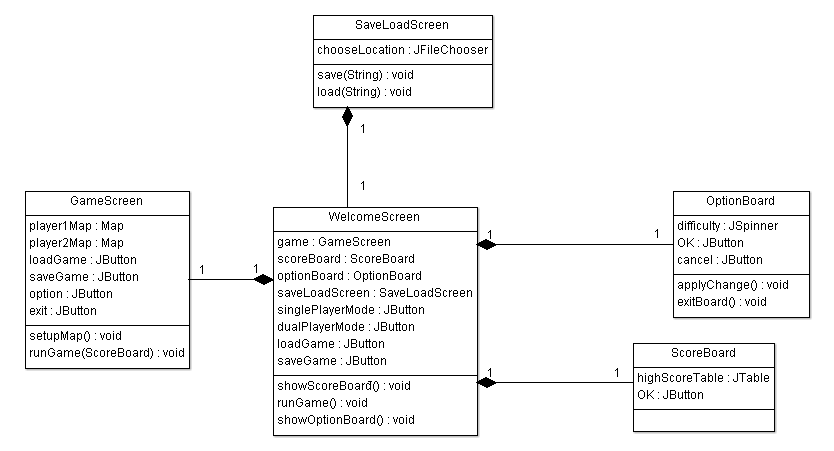
## UI Subsystem

The purpose of this subsystem is to present the user with a graphical user interface through which the game will be played. It will include:

* Main Page:
  + Play
    - Single Player
    - Multi-Player
  + Tutorial
    - Within tutorial we will have different slides to go over different basic control, where to look for basic functions such as hp bar, mp bar, saving point etc… and how the game will proceed.
  + Score Board
    - This will display the top 10 scores that have been achieved on the local machine.
  + Option
    - Sound Option
    - Display Option
    - Difficulties option



The subsystem will be composed of the following classes: Gamescreen, WelcomeScreen, SaveLoadScreen. OptionBoard, and Scoreboard. Each class represents a different page view of the game. The welcome screen is the central and the starting view to the game. From this screen, any of the other views mentioned above can be activated. If a user wants to switch between views they must pass through the welcome screen.



# Analysis

The user interface of the game must be intuitive and smooth. If the user interface is intuitive, people will have a more enjoyable experience and be more likely to play the game again. We must also avoid any possible confusing interfaces so to not aggravate the user. One can see in the UI subsystem that we created that there are clear menus in a logical hierarchy with limited amount of options which will assure a friendly UI. We will also make sure to use proper graphics meaning large distinct buttons with different color to make them visible so that navigating the game will be easy.

The game must have a proper refresh rate which means there is no “lag” with fluid graphics and user inputs. This means that the movement of the spaceship and objects respond to user inputs (change of speed, blowing up asteroid, etc…) nearly instantaneously. This once again will assure the quality and enjoyment of the game. We will implement this in the map class where there will be a timer refresh time. This refresh time will be adjusted through testing to assure a non lag graphic experience. We also have our physics subsystem which will be optimized to do calculations quickly so as to make sure changes are seen visually as fast as possible.

The game must also be challenging and unpredictable so that the user does not get bored. If the user is able to predict where the asteroids are going to be, then they will quickly become disinterested. Moreover the game must be able to properly handle network connectivity and audio capabilities. The proposed architecture will ensure that these requirements are met. We will implement this by making asteroids randomly placed when instantiated. This will mean a new experience after every game.

A separate subsystem for physics and graphics will be implemented. This will help us organize development and avoid the repetition of code. This will result in a smoother game because the physics and the graphics processes can be separated into their own thread. This division of physics may be changed since depending on how we instantiate objects we may have a more efficient dividing.

The elements that are most likely to change are the ones that are specific to the playable game itself. For example, the bonuses, levels, game difficulty, and alien ships will probably only be finalized later on in the software development. As we test our game, we will gain the ability to decide what playable elements would enhance the gaming experience. It is not very likely that our overall software architecture will change because its design is extensive and thorough.

# Design rationale

In Multiplayer mode, each player will have their own map object instance. After a player finishes their turn, we considered saving the positions of objects and score and restoring them at the end of the opponent’s turn. Instead, we chose to create 2 separate instances of the game map, and hiding the appropriate one during a player’s turn. This results in simpler programming, and no delay for loading and saving game states between turns.

There will be an options section that will be accessible from the main menu. This design decision was made in order to clean up the user interface of the program. In this sub-menu, there will be the options to change the difficulty and turn the sound on/off. By putting these options in a sub-menu rather than on the welcome screen, clutter is eliminated and the interface is made less confusing. Moreover, the decision was made to make the multiplayer option accessible after the “Play Game” button is pressed because we found this method to be the most intuitive. If the user wants to play a game, they will first have to go through the “Play Game” sub-menu to access the multiplayer game options because it is already assumed the player wants to play the game and not change game settings.

Also, originally the decision was made to create a separate “units” class that will contain all of the attributes that will affect the game objects (Such as position, speed, acceleration, etc...). However this class was too vague and grouped too many objects together. Instead we decided to implement an “Element” class with two sub-classes: PhysicsElement and GraphicsElement. Through this strategy, we can separate all of the methods that are related to game physics, and all of the methods that are related to the game graphics. This will make our software architecture much easier to follow as our project progresses.

# Workload breakdown:

Through the analysis of each team member's individual strengths we have divided the project accordingly:

**Neil Sabharwal:** UI Subsystem, Element Subsystem

**Hoai Phuoc Truong:** Object Subsystem, Element Subsystem

**Corey Perlman:** Documentation, Physics Subsystem

**Andrew Dennis:** Object Subsystem

**Jeffrey Leung:** Graphics Subsystem

**Richard Pringle**: Multiplayer Subsystem, Game Score Subsystem