ASTEROIDS ++ Software Architecture Document

Prepared by

260477045	Juan Guzman
260395716	Graham Ludwinski
260376364	Khoi Tran-Quang
260445956	Nicole Witter
260549690	Zhipeng Zheng

ECSE-321-001: Introduction to Software Engineering

Instructor: Professor Haibo Zeng

3/1/2013

VERSION 1.0

Contents

Revision History

Author	Date	Description	Version
Nicole Witter	*3/1/2013*	First version	<1.0>

1 Documentation Roadmap

1.1 Purpose and Scope of the SAD

This document serves to outline and describe the software architecture of the shooting game *Asteroids++*, which will be based off Atari's successful and popular retro game *Asteroids*, released in November of 1979.

The information is structured following a template developed at the Software Engineering Institute.

The intended audience of this document includes the software developers, the supervisors of this project, and the client base interested in the development of the game.

1.2 How the SAD is Organized

This SAD is organized into the following sections:

- **Document Roadmap and Overview:** provides information about this document and its intended audience. It provides the roadmap and document overview.
- **Architecture Background:** provides information about the software architecture. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. Contains detailed and specific information about the subsystems in the game.
- **Views:** includes diagrams to give readers an idea of the architecture.

- **Referenced Materials:** provides look-up information for documents that are cited elsewhere in this SAD.
- **Glossary and Acronym**s: has definitions of technical terms used in the documentation that may not be familiar to all readers.

1.3 Stakeholder Representation

Stakeholders of a software system are concerned with different aspects of the system. The following is a list of the stakeholders for *Asteroids++*, along with their apprehensions that may be dealt with in the document.

Stakeholder	Concerns
Users	Users will typically consist of kids, teenagers and adults. Concerns include whether or not the game is fun, is easy to understand, reliable and responsive.
Acquirers	Concerns include whether the system can be implemented on schedule.
Developers	*Asteroid++* developers are concerned with whether the user goals can be met, while remaining flexible.
Maintainers	A possible open source community will be concerned with if there are any reliable documentations in order to facilitate maintainability of the game.
Project manager	Concerns include whether or not the system can be implemented on schedule, if the architecture will allow team members to work independently and the correctness of the system.

1.4 Process for Updating this SAD

Discrepancies, errors, inconsistencies, and omissions in this document can be reported by filling out a form at www.asterods++.com under the Error Reports section of the site. The submitter will be notified when the developers have taken the case into consideration. Future updates will include the fix to the problem, along with the name of the person who informed the developers of the issue.

2 Architecture Background

2.1 Problem Background

2.1.1 System Overview

Asteroids++, as the name implies, shall be an improved version of the original *Asteroids*. The system will keep many aspects of the original game, but shall have new features which will greatly enhance the gameplay. A large variety of weapons, the possibility of having an alternate and more efficient method of controlling the ship, and an enhanced yet user-friendly graphical user interface will all be implemented. There will be different levels of difficulty, as well as appropriate sounds or music to enhance the gaming experience. Two players shall be allowed to play simultaneously. There will also be different saucers, the potential to save a game, and a possibility of online play. Also implemented will be a high score system that displays achievements and statistics. The goal of *Asteroids++* is to enable users to relive the glory of Atari's *Asteroids* in a different, but significantly improved light.

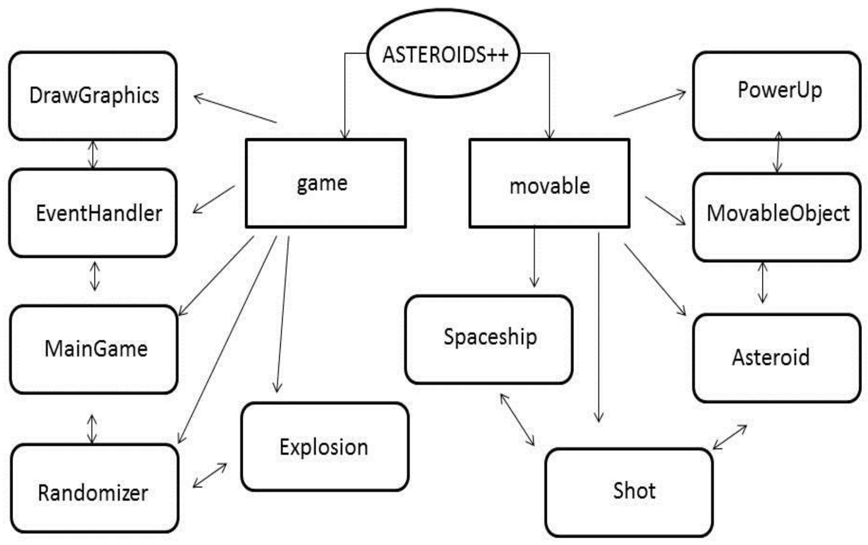

Figure 1: High-level view of the Asteroids++ Architecture

A high-level view of the Asteroids++ architecture is illustrated above. This illustration includes the main components of the system and the interactions between each other. The graphics subsystem must continuously communicate with the asteroids and spaceships subsystems as it will keep track of the positions of the objects in the game. The collision subsystem will react to any type of collisions that occur in-game. The sound subsystem will produce the sounds for the game. The options subsystem allows the user to modify the controls and other miscellaneous settings.

2.1.2 Goals and Context

The document provides a foundation on which the code for the system shall be based. The document will be referenced heavily in the implementation and testing phases of the software process. Proper architecture is a system framework that guides the development effort, while also reducing the cost and increasing the efficiency of the software. The architecture will fulfill the quality requirements. It will do so by creating a modular system that can be easily modified to incorporate the variety of requirements necessary to the successful implementation of the game. The two main packages will contain the different components, one with the “movable” aspects of the game, including MovableObject which encapsulates much of the necessary physics of movement in the game. The code is constructed in a way to lessen the CPU usage of a given system, therefore increasing and improving performance. Libraries will also be imported, as they reduce the code necessary to write thus making the system more maintainable, and flexible. In terms of the remainder of the quality requirements, the architecture handles these simply by running functional code. The design will such that the system will be able to run in any given time, making its availability very high and almost always. Its reliability is high as the system rarely crashes. It is compatible with Windows, Linux and Mac, as stated in the SRS. The event Handler class makes sure that the game runs with a PC which has a keyboard and mouse available. Design constraints force the developers to design the system in Java as majority of the team only know Java. A simple design will allow users to easily run the program, as mentioned in the SRS in the Quality

Requirements section. Stability of the game shall be very high due to error prone code and the nature of the code's architecture. Again, with less code, comes with easier maintainability and higher performance, reusability and flexibility.

2.1.3 Significant Driving Requirements

Asteroids++ is a game that:

- Aims to provide users with an exciting gameplay.
- Simple to get into.
- Allows users coming from any background to easily play.
- Has responsive controls.

2.2 Solution Background

2.2.1 Architectural Approaches (Design Rationale)

A shooting method that is separate from the ship method will allow for easier development. This design concept is more flexible and more organized as the shots are isolated entities from the ship. Power-ups will be more easily implemented. There will be less clutter, and it will be easier to read the code in general.

The architecture of the game will not be too complex and be simple. Both the developers and users must find it easy to use. The interface that the users will be presented with does not require much effort to play. No unnecessary navigation in-game will be required. Everything will be designed such that only the main or necessary things are required.

User accounts will not only enable users to save and load their game, but to record their gameplay statistics. This will also allow users to identify other users online.

The separation of our classes is meant to make our application much more modularized. By separating each functional unit into its own class, we can extend and inherit attributes from other classes to modify our game. This will for the easy addition of new features, game types, weapons, and ships. While somewhat simple, the application menu shall be designed for ease of use. When building it, we chose only what functions were absolutely necessary in order to allow the game to function properly. This is because in the context of our audience, people who want to play a simple game for a short period of time, most users don't care about more features. They simply want the least amount of friction when playing.

2.2.2 Analysis

Thorough testing will ensure that the game will run with minimal problems and attempt to reduce the number of bugs. The game may run on any operating system or environment provided that it is running on a decent computer and that Java is installed. The design shall be implemented in Java with the use of Java Swing. The game shall use an extremely simple interface which will only require very basic computer knowledge. Clear instructions must be included which will allow users to learn the gameplay mechanics in no more than a minute. The graphical user interface and the game in general must have a pleasant look and feel to it. The use of sprites and special types of animations may be a possibility. The controls shall be very responsive, such that there will be no delays, or extremely short delays (less than half a second) from keystrokes. Specific time delays may be used in the user interface to create special visual effects which serve to add a “cool factor” to the game.

The game must be designed in such a way that will enable the developers to easily cope with necessary changes. In other words, the design must be flexible and must allow for easy maintainability, and reuse. This is possible as the architecture of the game is not too complex and relatively simple. Clear and concise classes in the architecture design allow developers to quickly and readily pinpoint out their functionalities and properties or attributes. A class called Movable Object shall be implemented which will be used throughout the whole system, thus making the code extremely reusable. This class represents the superclass for any object that has a location, radius and velocity. This class is to be implemented as many of the entities in the game share the same attributes. Such a class will facilitate the coding and structure of the game. The game is written in Java, which is a pretty maintainable language. Getters and setters are also used to prevent unintended modification of class data. The developers will try to implement the minimum amount of code as possible, because a larger volume of code makes the system harder to maintain. Existing libraries will be used to increase the maintainability of the code.

Requirements that may change include the time delays used in the user interface. If it is perceived as annoying, the time may be either reduced or increased. The interface itself may be revamped numerous times for an even better look. Other things include the sprites and any visual features used. If a better concept for the general architecture is thought up, that will be implemented over the original one.

2.3 Software Subsystems

The Asteroids ++ video game will have many different subsystems to increase the modularity of the game. This video game has two major modules: Player Module and Sever Module. This modules will be formed by different subsystems. The player module will consist of all the necessary components that make up a working asteroids game.

Player Module:

- The Options subsystem will allow the users to change the controlling keys of the spaceship. Also, the user will be able to turn on/off the game sounds by using this subsystem. Mouse aim can also be enabled or disabled in this section. In the final version, this subsystem was no implemented.
- The Spaceship subsystem will provide moving capabilities to the spaceships. This subsystem will control the direction and acceleration of the spaceships depending on the user input or the artificial intelligence input. There will be two different types of spaceships that this subsystem will handle: player's spaceships and enemy's spaceships. In the final version, the movement of the spaceship was defined by the Moveable Object class, which represents an extremely versatile class that can control and define the movement, velocity, and location of the ship. The rotation of the ship is defined by the

changing angle of the ship. The wrapping of the ship in the screen is also controlled by the Movable Object Class.

- The Shooting subsystem will provide the shooting ability to the spaceship. This class will also manage and control any “special power” that the user acquires during the game (the implementation of “special powers” is optional; therefore, the functionalities of this class may not be exactly the same as described in the SAD when the product will be completed) It shall consist of a constructor which allows the developer to set the properties of the shots.
- The Asteroid subsystem will be responsible of controlling the physics involving asteroids and the behavior. This subsystem will control the speed and direction of any asteroid present in the game field. Also, this subsystem will be responsible of destroying and dividing asteroids whenever an asteroid is attacked by the users. Moreover, the asteroids subsystem will also equilibrate the difficulty of the game by adding or removing asteroids from the game field.
- The Artificial Intelligence (AI) subsystem will operate enemy’s spaceships. The main function of this subsystem is to attack player’s spaceships. This subsystem will operate two different types of spaceships. The first type of spaceships are “normal spaceships”, this spaceships are the weakest spaceships in the game. The second type are “bosses spaceships”, this spaceships will have more life (HP) and they will be able to use special weapons.
- The Collision subsystem will be dedicated to detect any collision of asteroids with spaceships and bullets with asteroids. Numerous methods will be used to detect collisions

between the entities. Within these methods are what activate the animations for the explosions and other properties relating to explosions.

- The Graphics subsystems will render all the graphics of the game. This subsystem will be responsible of drawing the user interface. Also, this game will continuously update the user interface when the users are playing to create the game animations. It will consist of methods which draw the ship, shots, and asteroids. Using affine transform, special effects such as translation and rotation may be implemented. Sprites will be initialized in this class, and the display of specific information in the form of strings will be declared here.
- The Sound Subsystem will manage the sounds of the game. This subsystem will control the volume of the sounds and the playlists of the game. In the final version of the code, this was all implemented in the main game class, as errors resulted when the sounds were placed in a separate sound class.

Server module:

- The Server subsystem will be responsible of handling connections between different players, and maintaining (updating) the database.
- The Highest Scores subsystem will load and display the users records stored in the database.

Interface:

Only the Player subsystem will use a user interface. The size of this interface will be 1024x768 pixels. The main interface will have different buttons to allow the users to create an account, log into an existing account, start the game (single player or multi player), and access the option menu. When the users create or log into an account, they will be restricted to use alphanumerical characters only.

The option menu interface will have different icons to allow the user to turn on/off the audio and change the controlling keys of the spaceship. The input in this interface will be restricted to alphanumerical characters.

The highest score interface will simply show the highest scores achieved by the players in single player or multiplayer.

The game interface will have a static background. The asteroids, the enemy spaceships and the user spaceship will have dynamic animations. If the asteroids are bigger than the default size, they will break in different pieces. This interface must be able to recognize and react to the user interface. Also, this interface will only react to specific key combinations selected by the user in the options menu.

Examples of interfaces:

Main screen:

Game Background:

3 Views

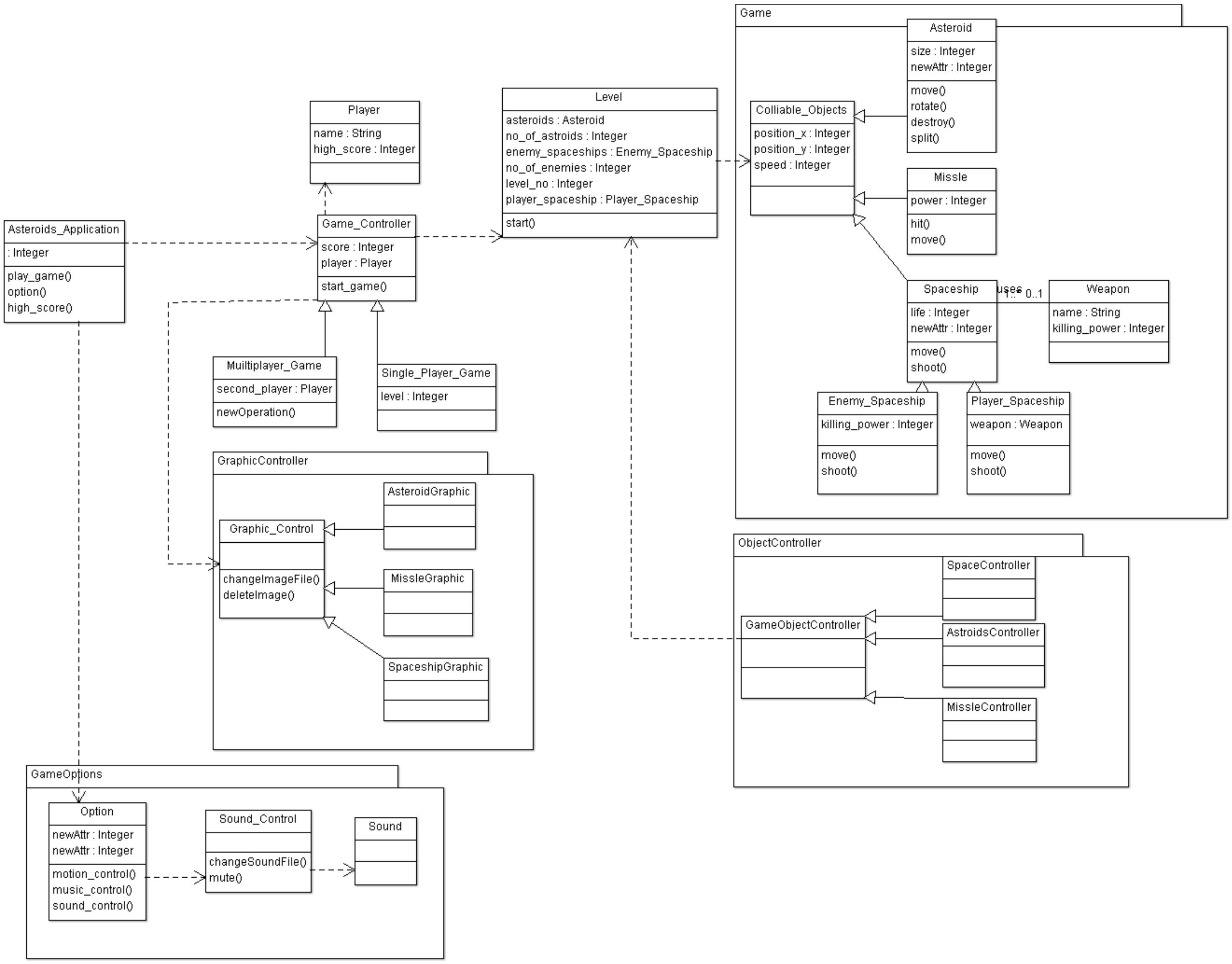

Figure 2: General Class Diagram of Asteroids++

IMPLEMENT PACKAGE DIAGRAMS HERE

4 Referenced Materials

Barbacci 2003	Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs)*, Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html>.
Bass 2003	Bass, Clements, Kazman, *Software Architecture in Practice,* second edition, Addison Wesley Longman, 2003.
Clements 2001	Clements, Kazman, Klein, *Evaluating Software Architectures: Methods and Case Studies,* Addison Wesley Longman, 2001.
Clements 2002	Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Staf-ford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2002.
IEEE 1471	ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Ar-chitectural Description of Software-Intensive Systems*, 21 Sep-tember 2000.

5 Directory

5.1 Glossary and Acronym List

Term	Definitions
Software architecture documentation (SAD)	A complete description of the behavior of a system to be developed and may include a set of use cases that describe interactions the users will have with the software. (1)
Software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
View	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
View packet	The smallest package of architectural documentation that could practically be given to a stakeholder. The documentation of a view consists of one or more view packets.
Viewpoint	A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.
Artificial intelligence (AI)	Used to produce the illusion of intelligence in the behavior of non-player characters.