Super Asteroids (DRAFT)

Contents

[1. Introduction 3](#_Toc464554894)

[2. Application Overview 4](#_Toc464554895)

[2.1. Startup 5](#_Toc464554896)

[2.2. Importing A Game Configuration 6](#_Toc464554897)

[2.2.1. Game Configuration Content 6](#_Toc464554898)

[2.3. Starting A Game: Ship Building 7](#_Toc464554899)

[2.4. Starting A Game: Gameplay 7](#_Toc464554900)

[2.4.1. Interface 7](#_Toc464554901)

[2.4.2. Rules 7](#_Toc464554902)

[2.5. Quick Play 7](#_Toc464554903)

[3. High Level Design 8](#_Toc464554904)

[3.1. The Data Importer 8](#_Toc464554905)

[3.2. The Ship Builder 9](#_Toc464554906)

[3.3. The Game Engine 9](#_Toc464554907)

[4. Low Level Design 9](#_Toc464554908)

[4.1. The Data Importer 9](#_Toc464554909)

[4.1.1. Game Configuration Data: JSON 9](#_Toc464554910)

# Introduction

This document serves as the software specification for the Super Asteriods program developed in the CS240 (Advanced Programing Concepts) course at BYU. The specification is presented in three sections that describe the software in increasingly technical detail – beginning with an operational (or user) view, and progressing to low-level design documentation. This document is intended to enable a student/developer to quickly understand the form and function of Super Asteroids as it exists, and enable them to extend and modify the software as required.

# Application Overview

Super Asteroids is an Android application inspired by the hit arcade game “Asteroids” released in 1979.

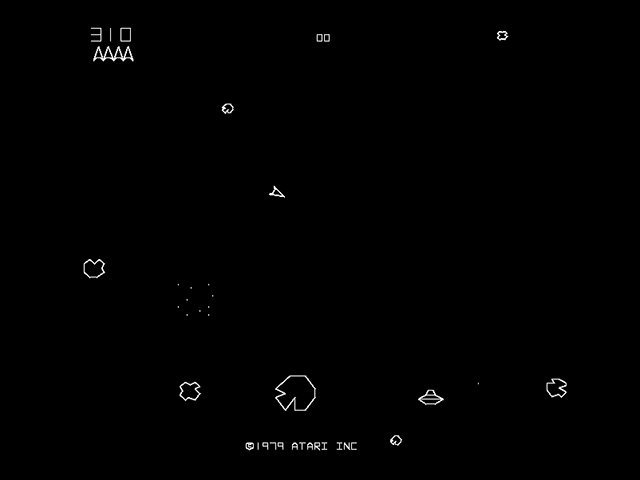


Figure : “Asteroids”

*https://upload.wikimedia.org/wikipedia/en/1/13/Asteroi1.png*

As in the original, players of Super Asteroids must navigate a ship through asteroid infested space, “blasting” their way from level to level. Unlike the original, Super Asteroids takes advantage of modern mobile computing technologies - including high resolution color graphics, and a touchscreen interface.



Figure : "Super Asteroids"

In addition to these gameplay advancements, Super Asteroids introduces a new dimension to the game by enabling users to vary gameplay through dynamically loaded game configurations (see section for more detail). The following subsections explain the processes of configuring and playing the game.

# Startup



Figure : Super Asteroids Startup Screen

Upon launching the Super Asteroids application, the user is presented with the startup screen shown in . From here the user can select “Start Game”, “Quick Play”, and “Import Data” (via “tapping” of the desired button).

**Start Game:** Launches the “ship builder” as part of play, this allows the player to choose the principle components of their spaceship; once a ship is fully designed the player can then choose to start playing.

**Quick Play:** Generates a random/default ship and starts the game.

**Import Data:** Brings up a list of available game configuration data files.

**Note:** The Start Game and Quick Play buttons will not operate if the internal database has not been populated via the Import Data screen. (Could/Should they be visibly disabled?)

presents the navigational relationships between the Start Screen, Ship Builder, Data Importer, and Game Screen states of Super Asteroids.

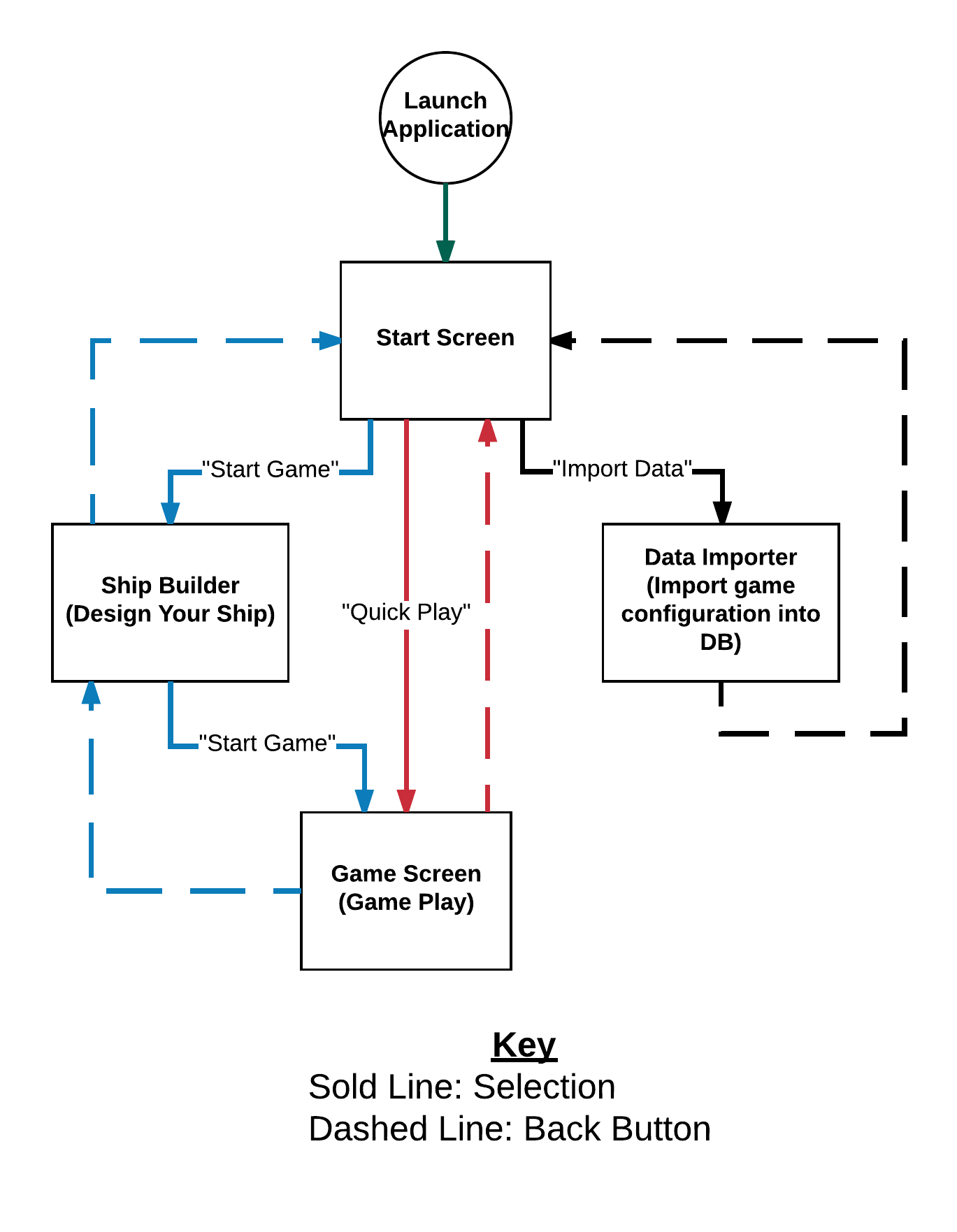
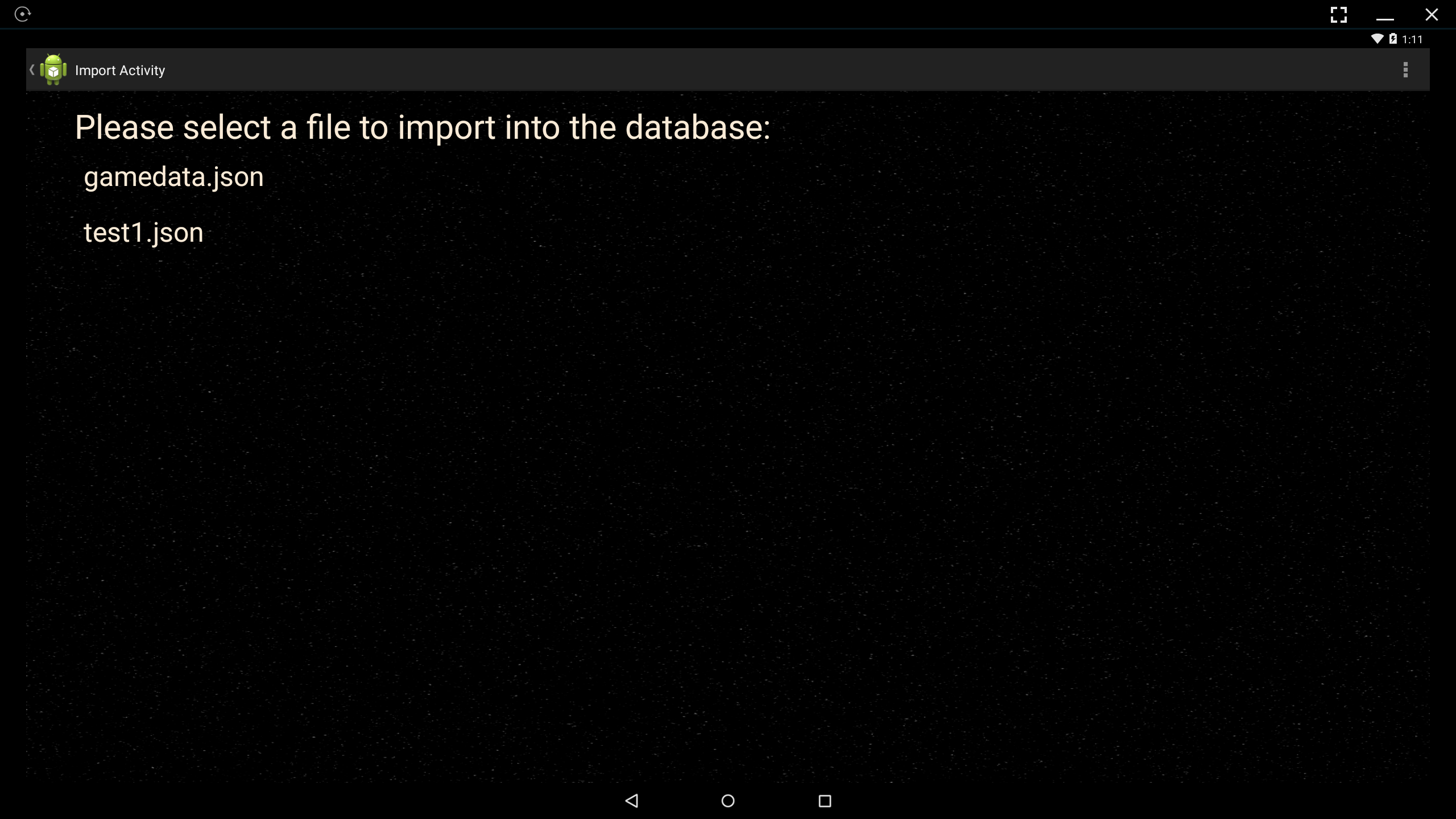


Figure : Super Asteroids Program Flow

# Starting A Game: Importing A Configuration

Tapping the Import Data button brings up the Data Importer screen. Here a user is presented with a list of available game configuration files. Tapping a filename from this list imports the corresponding file’s contents into the game’s internal database. Gameplay is based upon the current content of this database. Once the desired game configuration has been loaded the user can navigate back to the startup screen.



Note: If a user taps more than one configuration file the latter will override the former. In other words, the most recent (last) configuration selected is the one that will be available during gameplay.

# Game Configuration Content

Each game configuration file describes the elements of game play that will be presented to the player. These include: Number of levels, level content (i.e. background imagery, number and type of asteroids spawned at level start, level music etc.), ship configuration options, and asteroid configuration options (selected for each level). When a file is selected, its contents are read into the application’s internal database *replacing* any configuration data previously present. When a level is started, its description is loaded from the database and rendered to the screen.

Certain elements of a level are static and have predetermined sizes and positions that must be indicated in the game configuration. These elements comprise the “background objects” of the game and provide interesting scenery during gameplay (i.e. planets and space stations etc.).

Super Asteroids also includes moving objects, such as asteroids (and possibly space ships) that travel around the level map during gameplay. These objects are randomly assigned positions and velocities at the beginning of level-play, and their position, orientation (and possibly size) are updated dynamically at runtime. Both they type, and number of each type, of asteroids is specified for each level in the game configuration.

**Note:** At the beginning of a level (during asteroid placement) asteroids can NOT be placed on top of the player’s ship. A “safety zone” around the ship’s starting location (i.e. the center of the level) prevents asteroids from initially spawning too close.

Currently there are three types of asteroids available in Super Asteroids. Each of these behaves differently when destroyed (i.e. when the asteroid runs out of hit points).

1. Regular Asteroids (static size): Break into two smaller fragments with ½ the diameter (scale) of the original asteroid.
2. Growing Asteroids (grow slowly over time): Break into two smaller (growing) fragments each initially ½ the diameter (scale) of the original asteroid.
3. Octeroid Asteroids (static size): Break into eight smaller fragments, each 1/8th the diameter (scale) of the original asteroid.

For all three asteroid types, all fragments begin with ½ the hit points that their “parent” asteroid had (i.e. they are half as strong as the asteroid they broke away from). An asteroid splits once or twice before being completely destroyed (i.e. disappearing vs splitting).

By default, the user’s ship (which is discussed in greater detail in the next section) is initially located at the center of a level, and can then be moved during gameplay.

Section 3.1 discusses the organization and content of configuration files in greater detail.

# Starting A Game: Ship Building

If the desired game configuration, including game levels, asteroids, ship parts etc., have already been loaded into the game’s database (possibly during a previous use of the program), then a user can proceed directly to the ship building phase of game play by tapping the “Start Game” button on the Startup Screen. This will launch the ship builder.

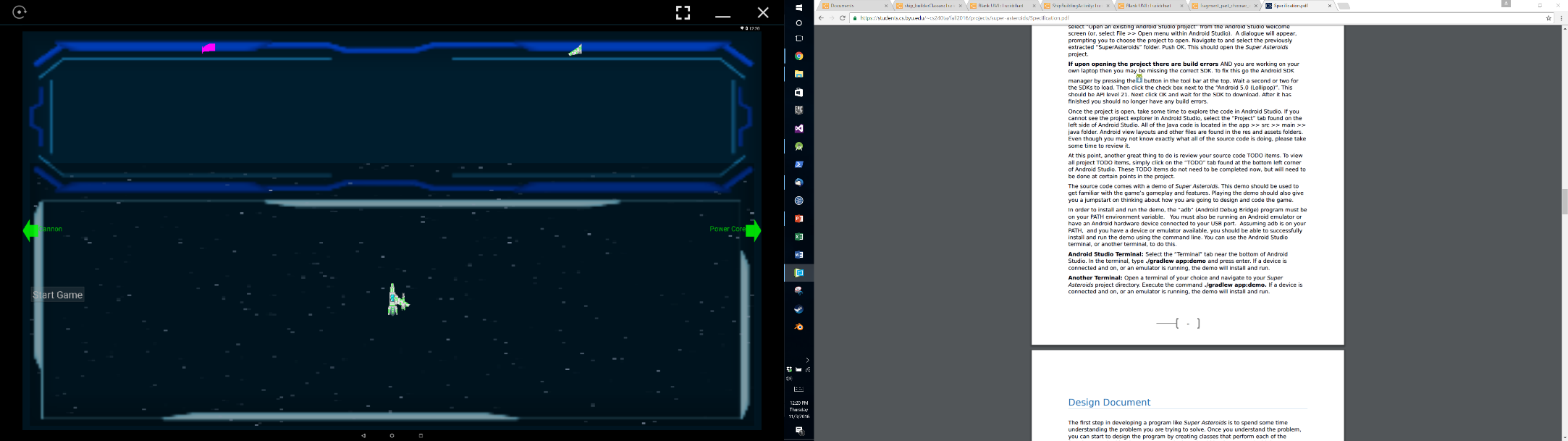


Figure 5: The Ship Builder

The ship builder is comprised of five screens that present the player with options from which to construct their ship. These options are derived from the configuration information currently present in the game database. Each screen offers the user a choice between components of the same type - enabling a large number of possible ship configurations.

The player navigates from screen to screen by swiping left, right, up, or down as prompted by the green arrows located on the edges of the screen. The specific navigational “flow” between these screens is implementation specific and must be determined by the developer. Figure 6 present two possible ways that the screens of the ship builder may be organized.

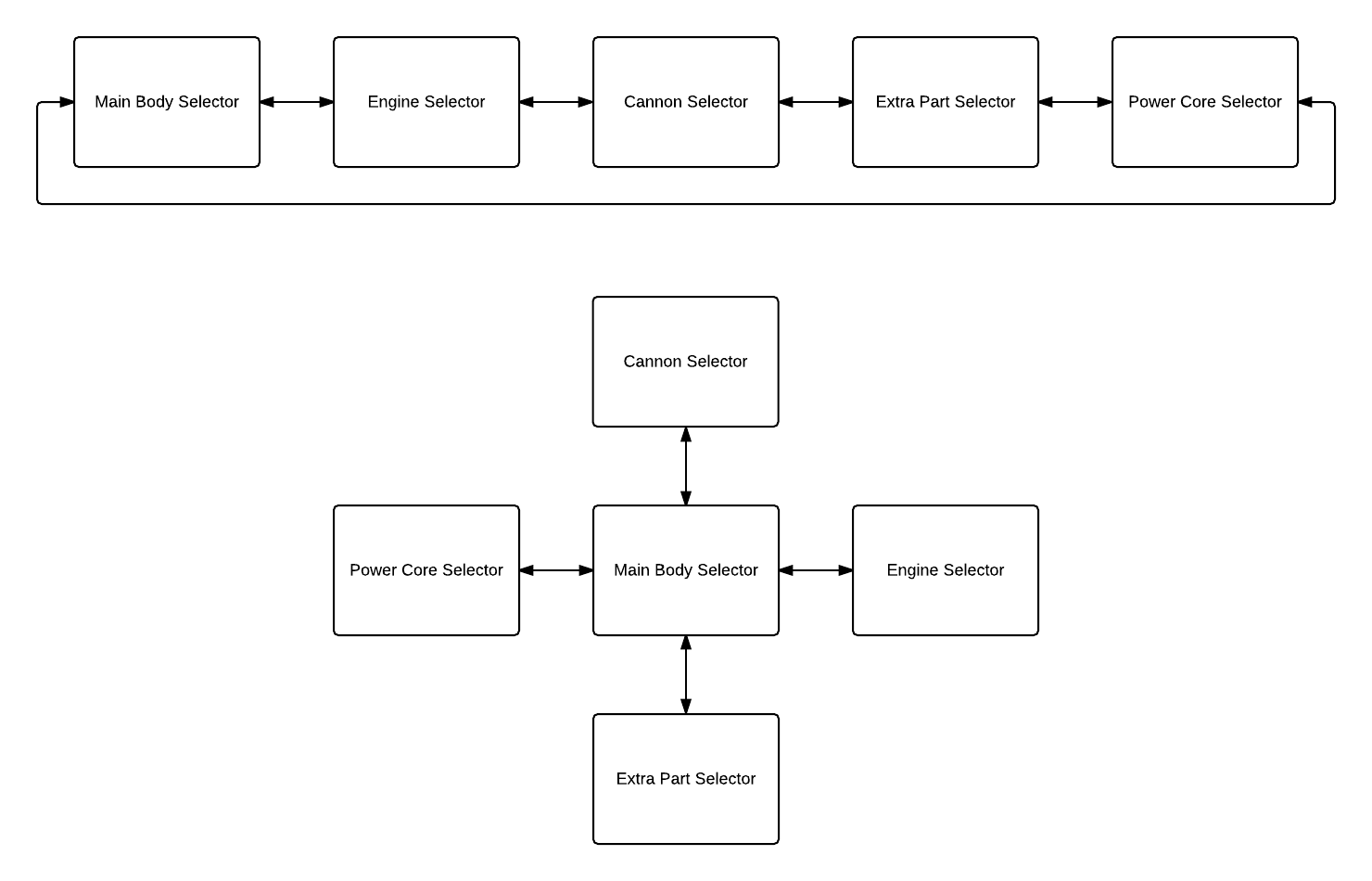


Figure 6: Navigating The Ship Builder

When the Ship Builder is first launched, and no ship parts have been selected, the “Start Game” button is disabled – preventing the player from attempting to start the game with an incomplete ship. Once selections for all five ship parts have been made, the button becomes enabled and clicking it will cause gameplay to begin.

# Starting A Game: Gameplay

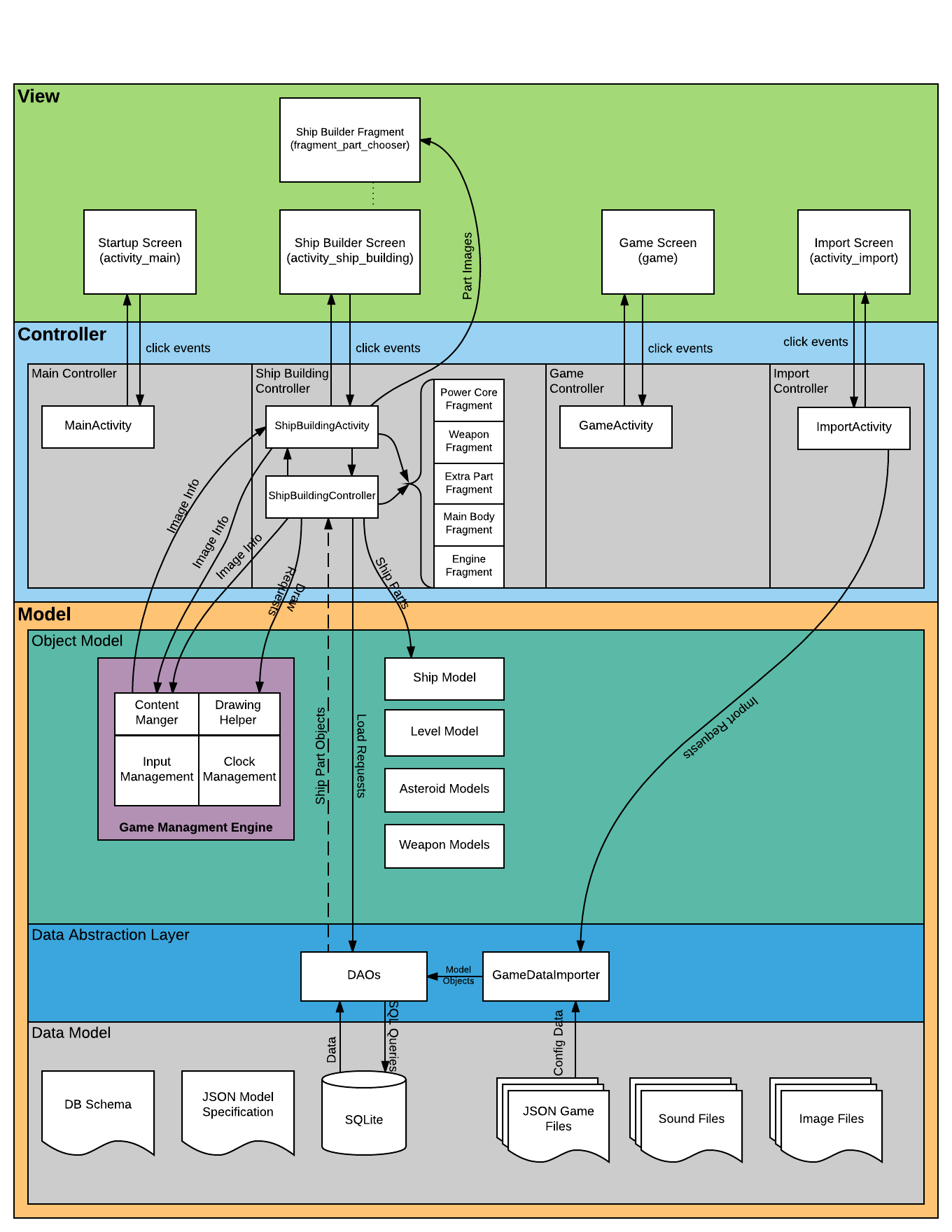
# Interface

# Rules

# Quick Play

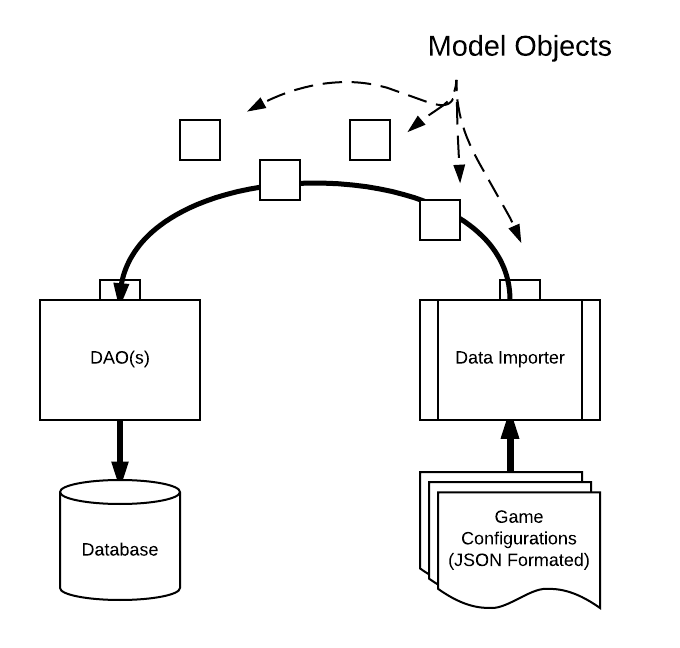
The “Quick Play” button of the startup screen offers users the ability to jump right into the action. If the desired game configuration (game levels, asteroids, ship parts etc.) has already been loaded into the game’s database (possibly during a previous use of the program), then pressing this button will allow the user to skip the ship building phase and immediately begin playing with a default ship configuration…

# High Level Design DRAFT



# The Data Importer

The data importer is responsible for loading game configuration files into the program’s internal SQLite database. This function is initiated when the player clicks on a filename on the Data Importer screen.



The game engine utilizes the contents of the database to determine the details of gameplay. Game configuration files are organized according to the structure represented in Figure 7. (A detailed specification of the JSON interface represented here can be found in section 4.1.1)

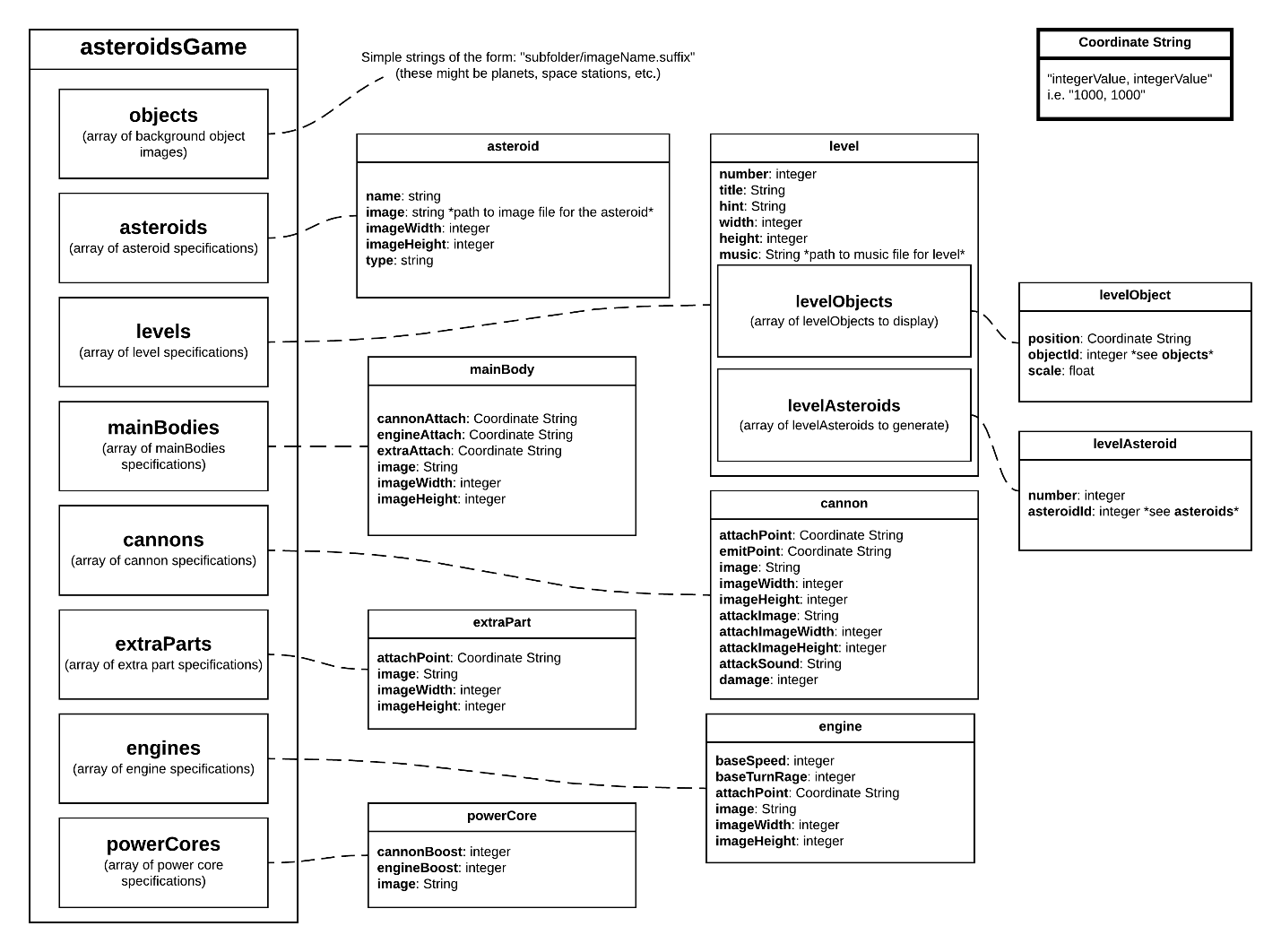


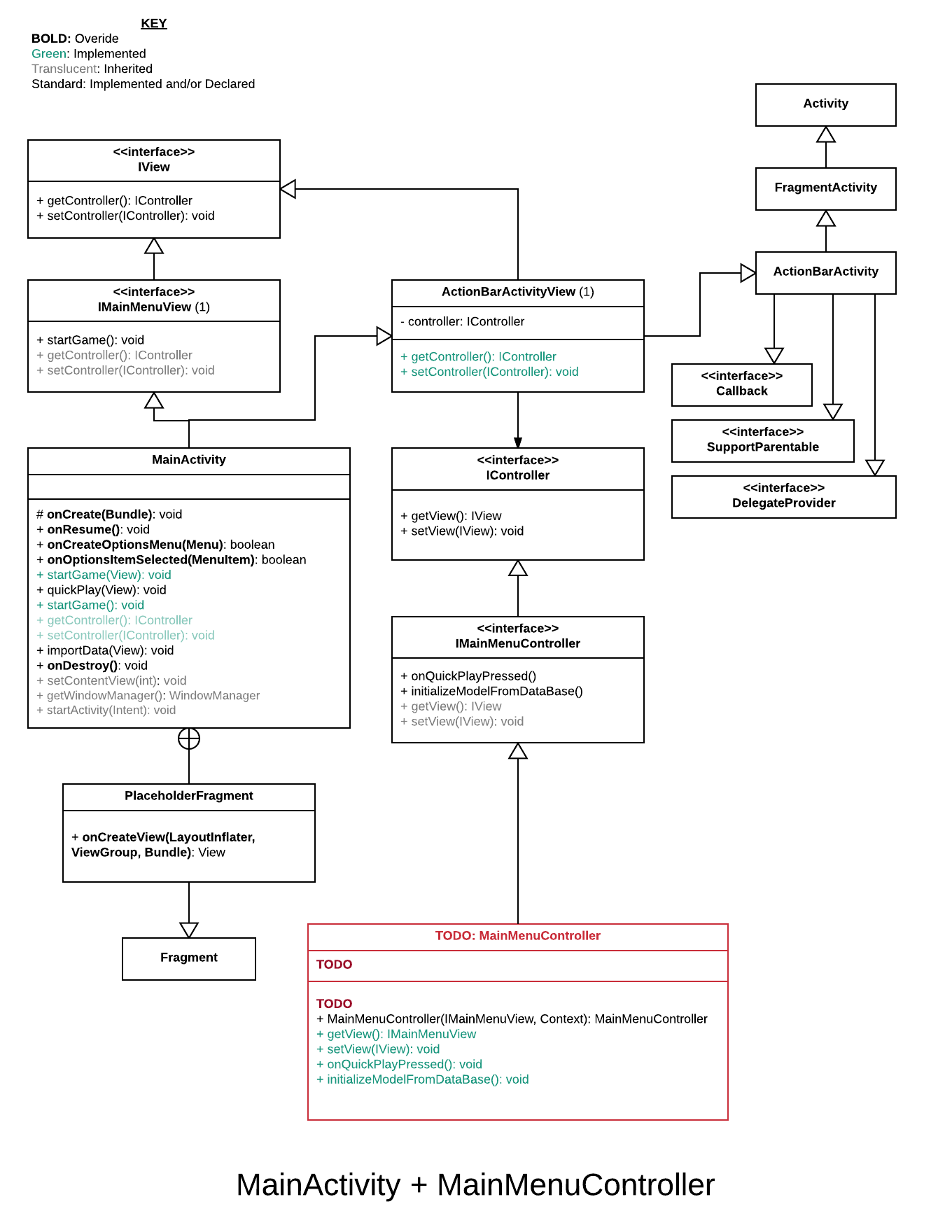
Figure : Super Asteroids JSON data model

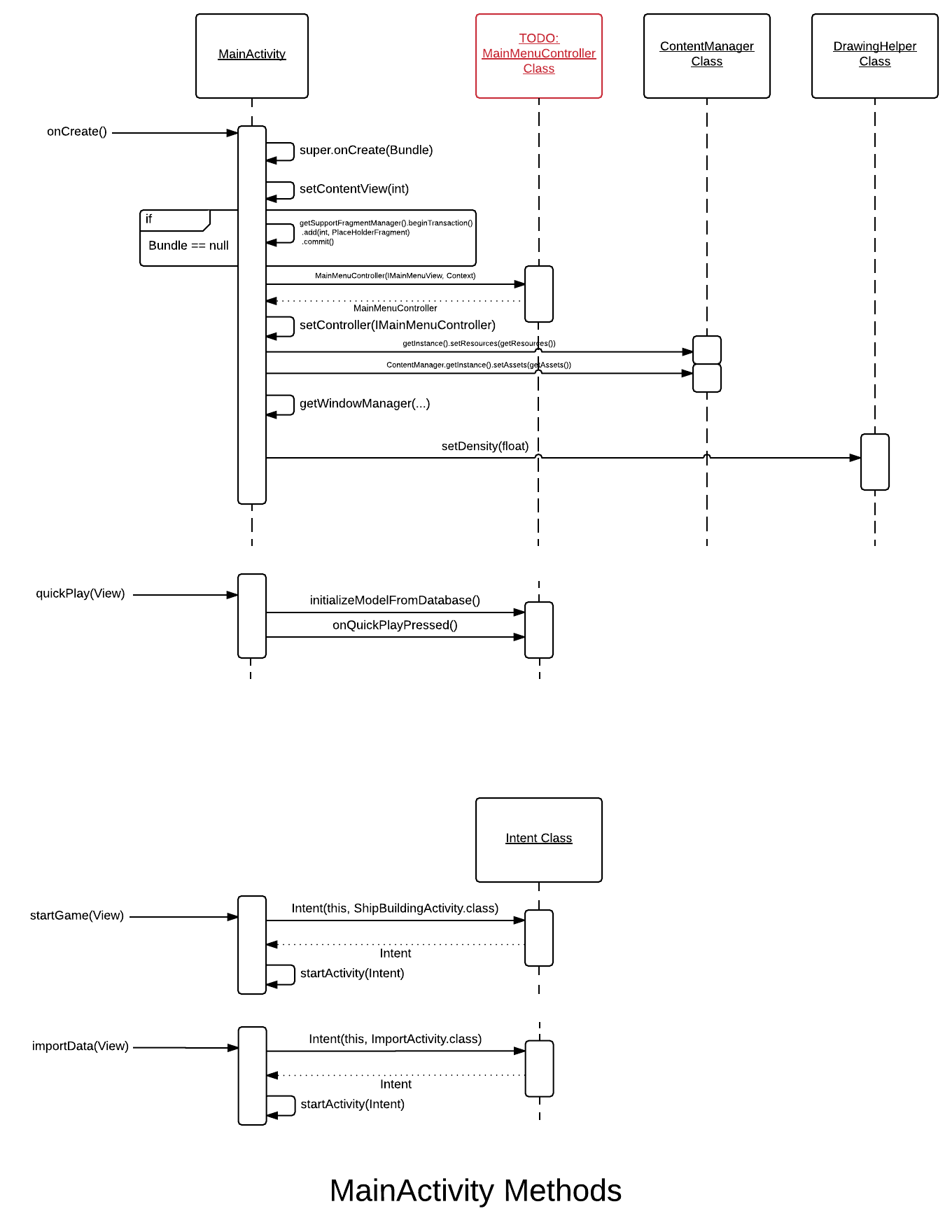
The data importer parses the data objects represented in such a file into representative model objects. These objects are then stored in the program’s internal database for retrieval at game play time.

# The Ship Builder

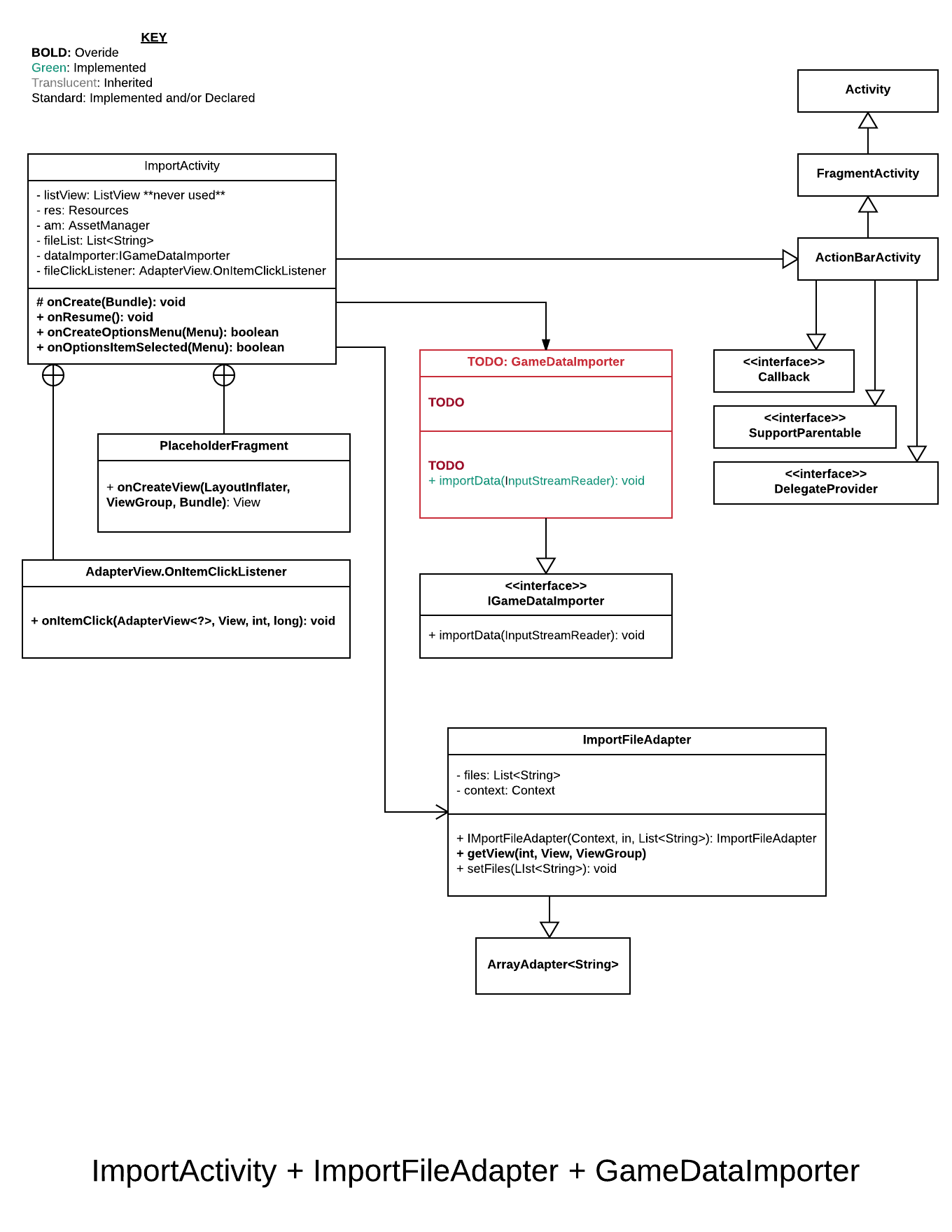
# The Game Engine

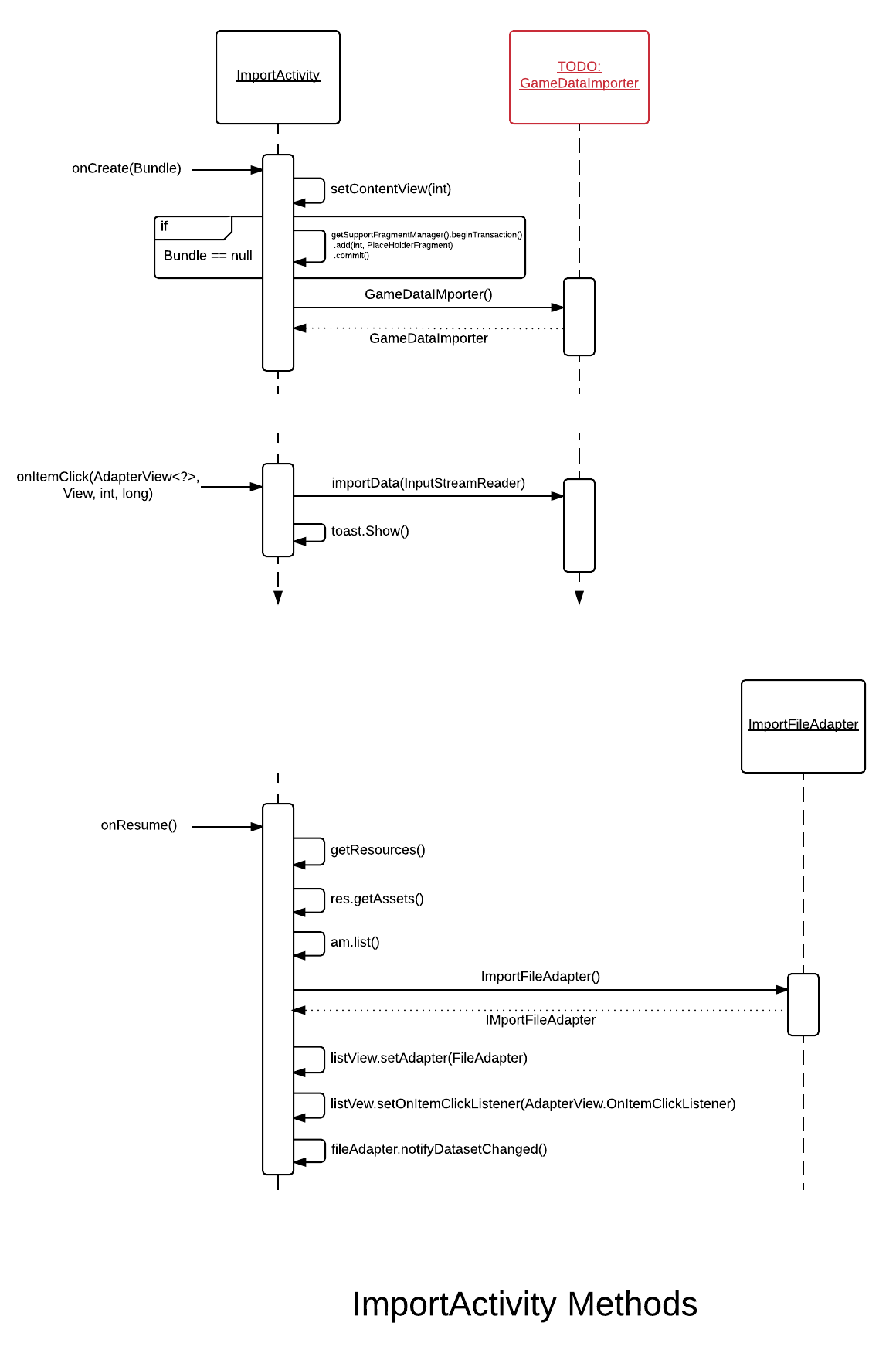
# Low Level Design





# The Data Importer





# Game Configuration Data: JSON

Game configurations for Super Asteroids (including custom levels, imagery, etc.) can be specified via an Asteroids game data file (JSON format). These files utilize the JSON data exchange language per the following specification.

***Note:*** *Any strings representing file paths subscribe to the following conventions:*

• All files exist in the assets folder, or some sub folder of the assets folder.

• File path strings do not start with a file separator and do not contain the word “assets”.

*Example: If the file is in the images folder, which is in the assets folder, the path should be “images/planet0.png”.*

***Note:*** *The “Coordinate String” type is defined as a string containing an x coordinate and y coordinate separated by a comma.*

*Example: “100,100”.*

***Note:*** *Italicized text indicates a comment below*

{

“asteroidsGame”: { *This object holds all of the data for the Asteroids game configuration.*

“objects”: [ *An array of strings. These strings represent the path to the image file for a background object. Can be empty.*

],

“asteroids”: [ *An array of Asteroid Types. Should not be empty.*

{Asteroid Type: Contains information describing an asteroid type.

“name”: String, The name of the asteroid type.

“image”: String, The path for the image file for the asteroid.

“imageWidth”: integer, The pixel width of the asteroid’s image.

“imageHeight”: integer, The pixel height of the asteroid’s image.

“type”: String The type of the asteroid. This is used to determine the behavior and characteristics of the asteroid.

},…

],

“levels”: [An array of Levels. Should not be empty.

{Level: Contains information describing a level.

“number”: integer The level number.

“title”: String The level title.

“hint”: String The level hint to be displayed with the title.

“width”: integer The pixel width of the level.

“height”: integer The pixel height of the level.

“music”: String The path to the music file to be played with the level. “levelObjects”: [*An array of Level Objects. Can be empty*

{Contains information describing a level background object.

“position”: Coordinate String. The position in the level to draw the object.

“objectId”: integer. The ID of the object to draw. IDs corresponds to the ordering of objects in the objects array.

“scale”: float The scale to draw the object at.

},…

“levelAsteroids”:[ *An array of Level Asteroids. Should not be empty.*

{Level Asteroid: Contains information describing the asteroids in a level.

“number”: Integer The number of asteroids of this type to generate at the beginning of

the level.

“asteroidId”: Integer The ID of the asteroid type to generate.

},…

]

},…

“mainBodies”: [*An array of Main Body objects. Should not be empty.*

{Main Body Object: Contains information describing a main body part of the ship.

“cannonAttach”: Coordinate String The point on the main body image where the

cannon should be attached.

“engineAttach”: Coordinate String The point on the main body image where the

engine should be attached.

“extraAttach”: Coordinate String The point on the main body image where the

extra part should be attached.

“image”: String The path to main body image.

“imageWidth”: Integer The pixel width of the main body image.

“imageHeight”: Integer The pixel height of the main body image.

},…

]

“cannons”: [ *An array of Cannon objects. Should not be empty.*

{Cannon Object: Contains information describing a cannon part of the ship.

“attachPoint”: Coordinate String The point of the cannon image that attaches to the

main body image.

“emitPoint”: Coordinate String The point of the cannon image the projectile is emitted from.

“image”: String The path to cannon image.

“imageWidth”: Integer The pixel width of the cannon image.

“imageHeight”: Integer The pixel height of the cannon image.

“attackImage”: String The path to the cannon’s projectile image.

“attackImageWidth”: Integer The pixel width of the cannon’s projectile image.

“attackImageHeight”: Integer The pixel height of the cannon’s projectile image.

“attackSound”: String The path to the cannon’s projectile sound file.

“damage”: Integer The base damage for each projectile.

},…

]

“extraParts”: [ *An array of Extra Part objects. Should not be empty.*

{Extra Part Object: Contains information describing an extra part of the ship.

“attachPoint”: Coordinate String The point of the extra part image that attaches to the main body

image.

“image”: String The path to extra part image.

“imageWidth”: Integer The pixel width of the extra part image.

“imageHeight”: Integer The pixel height of the extra part image.

},…

]

“engines”: [ *An array of Engine objects. Should not be empty.*

{Engine Object: Contains information describing an engine part of the ship.

“baseSpeed”: Integer The base maximum velocity of the ship in pixels per second.

“baseTurnRate”: Integer The base turn rate of the ship in degrees per second.

“attachPoint”: Coordinate String The point of the engine part image that attaches to the

main body image.

“image”: String The path to engine part image.

“imageWidth”: Integer The pixel width of the engine part image.

“imageHeight”: Integer The pixel height of the engine part image.

},…

]

“powerCores”: [ *An array of Power Core objects. Should not be empty.*

{Power Core Object: Contains information describing a power core part of the ship.

“cannonBoost”: Integer The value of extra damage that should be added to the cannon’s

base damage.

“engineBoost”: Integer Adds to the base speed of the engine.

“image”: String The path to the power core image.

},…

]

}

}

# The Ship Builder

