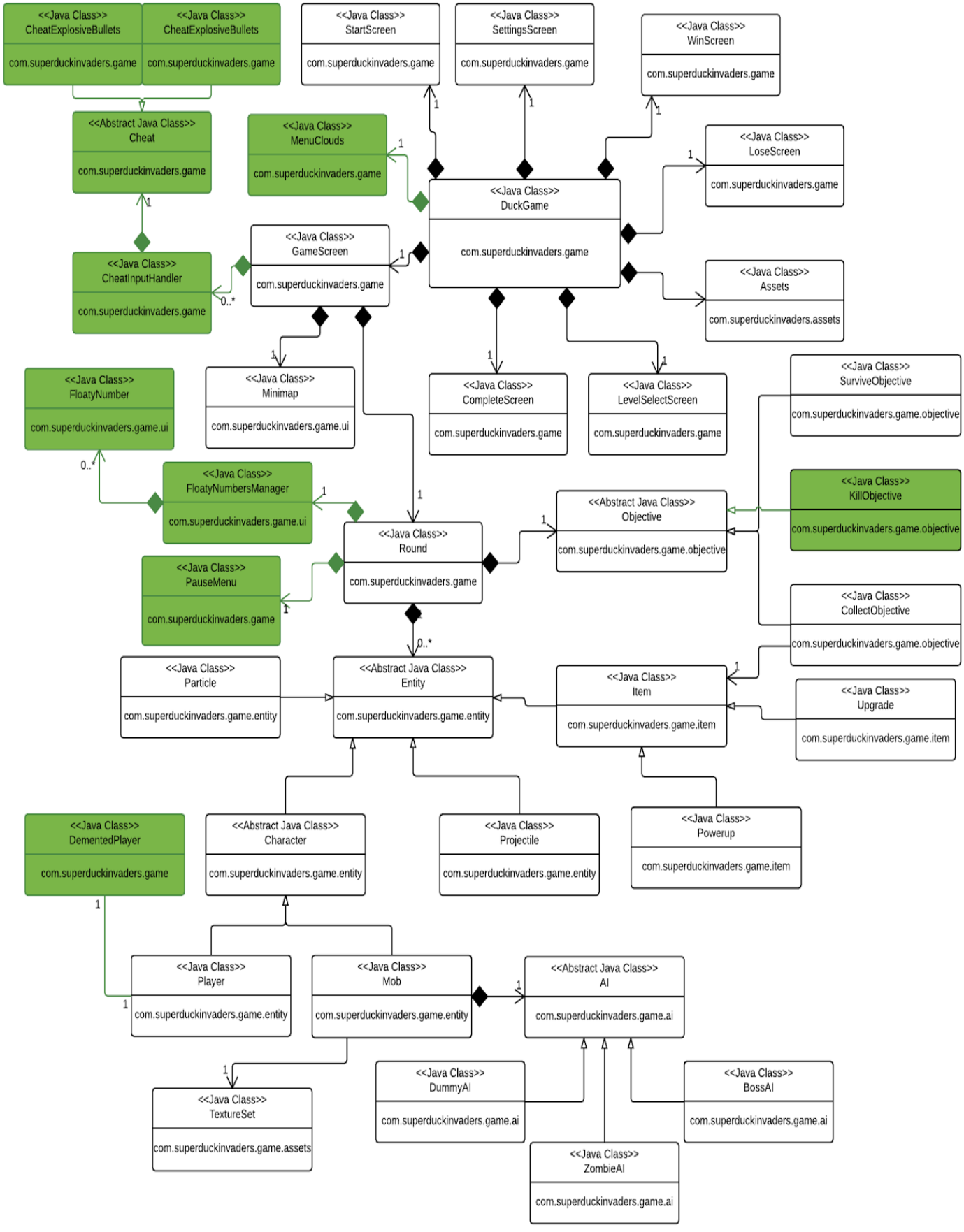
Assessment 4 Architecture Changes

Architecture largely stayed the same and small additions and tweaks were made as necessary to meet our requirements.

ID	Affected class(es)	Explanation
AR1	Cheat, CheatInputHandler, (PauseMenu)	Cheat is an abstract class which should be extended by classes, this allows the CheatInputHandler class to call functions of each Cheat subclass such as activate. One instance of a Cheat subclass is associated to each handler instance and there are multiple handler instances in the PauseMenu class (AR4)
AR2	CheatExplosiveBullet, CheatBouncingDuck	Both inherit from the Cheat class and relate to requirement G15. These subclasses are used to implement specific cheats.
AR3	DemenetedPlayer, Player	The DemenetedPlayer class contains all the methods and variables to handle when the player should become demented and how it should behave. This class keeps stores the Player instance which it is affecting. Relates to requirement G14.
AR4	PauseMenu	The addition of the PauseMenu class was required firstly to allow a pause state in which the player could safely enter cheat codes and so relates to requirement G15. The secondary function of the PauseMenu is to allow the player to transition from the game screen to the start screen or to quit the game. A PauseMenu is instantiated and kept within the the round class. Relates to requirement G15
AR5	MenuClouds	Addition of MenuClouds class which handles the generation and rendering of floating cloud sprites on the various menu screens. Relates to requirement S2 specifically the aesthetics. MenuClouds is part of DuckGame instead of each of the menu screen classes since we wanted the state of the clouds to persist throughout the various menu screens.
AR6	FloatyNumber, FloatyNumberManager, AnimatedText	FloatyNumber and FloatyNumberManager classes were added as a replacement of AnimatedText. Splitting the AnimatedText class into two classes made the class easier to modify and get the behaviour we wanted. FloatyNumberManager has multiple FloatyNumber instances which it updates and renders. Relates to requirement S2
AR7	KillObjective	The addition of a new Objective subclass, this change was not made in response to meeting any requirements except to make the game more fun S2. No architectural modification beyond the introduction of this class were made.

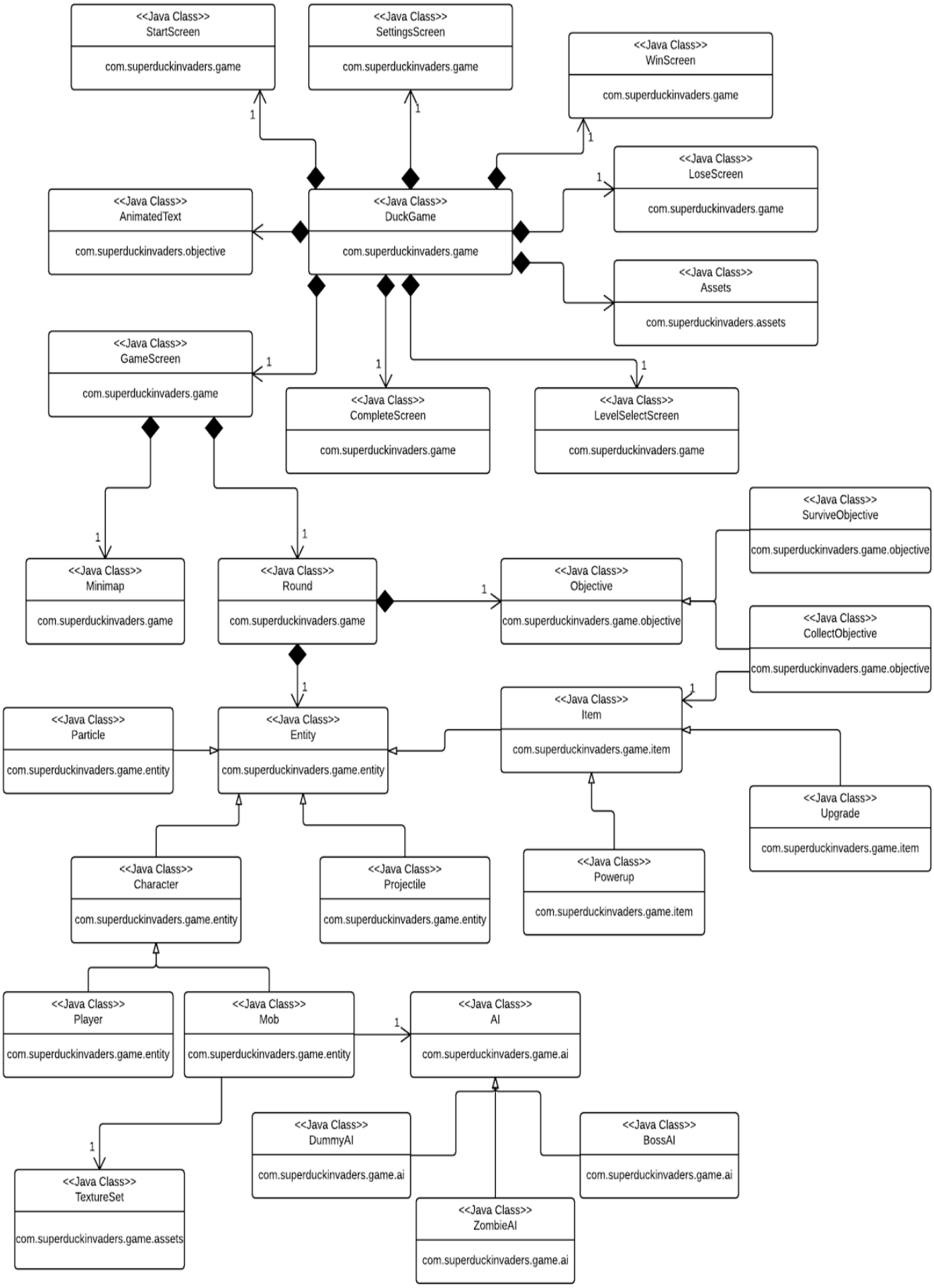
Assessment 4 UML Diagram



Assessment 3 Architecture Changes

- Addition of a LevelSelectScreen as a screen which can be accessed from the StartScreen. This change was made to meet requirement G1. Like other screens LevelSelectScreen inherits from the LibGdx Screen class.
- CompleteScreen was added to signify the end of the game and was made to meet requirements S4 and S2. Also inherits from LibGdx Screen Class.
- Addition of a SettingsScreen which was added to control volume, other settings can be added later also inherits from LibGdx Screen class.
- Addition of a SurviveObjective to the architecture to meet the requirements of two different types of objectives G2. This class is similar to the CollectObjective in that it inherits from the Objective class.
- Addition of BossAI to allow different types of enemies and obstacles, requirement G2 and S2 since the variety makes the game more attractive. Simlar to ZombieAI and DummyAI, BossAI inherits from AI.
- Addition of a Minimap class. An instance of this is kept inside the GameScreen where it is used to render a working Minimap graphic. Requirement I4.
- Addition of AnimatedText kept in the DuckGame class used to render and generate animated text.

Assessment 3 Architecture UML diagram



UML and Lucidchart

The above UML 2.0[1] diagrams illustrate the concrete architecture of the initial implementation of our game. UML 2.0 was chosen as it is a well-defined standard way of expressing software architectures, and as such using it likely negates the need for any future maintainers of our code to learn another modelling language in order to understand the diagrams. The class diagram shows how the concrete Java classes in our game are related, including how they inherit from each other (*"is-a"* relationships) and where classes are composed of other classes (*"has-a"* relationships). The state diagram shows a representation of the different screens a user could be presented with in the game, and what is required to transition between them. The activity diagram gives a high-level representation of the operation of our game, and what happens when the victory/ loss conditions are met.

All diagrams were created using an online chart drawing tool called Lucidchart[2]. Lucidchart supports a large range of UML symbols, as well as other notations.

Any associations or classes highlighted in green have been added after assessment 3 for assessment 4 this only included on the assessment 4 diagram on the following page.

Bibliography

[1] Object Management Group, "Unifed Modeling Language: Infrastructure", 2006. [Online]. Available: http://doc.omg.org/formal/2005-07-05.pdf. [Accessed: 12- Feb- 2016].

[2] Lucidchart, "Flow Chart & Diagram Maker - Lucidchart". [Online]. Available: http://www.lucidchart.com/. [Accessed: 10- Feb- 2016].