Architecture

Group 8 - Delta Ducks

Yumis Zyutyu

Zac Challis

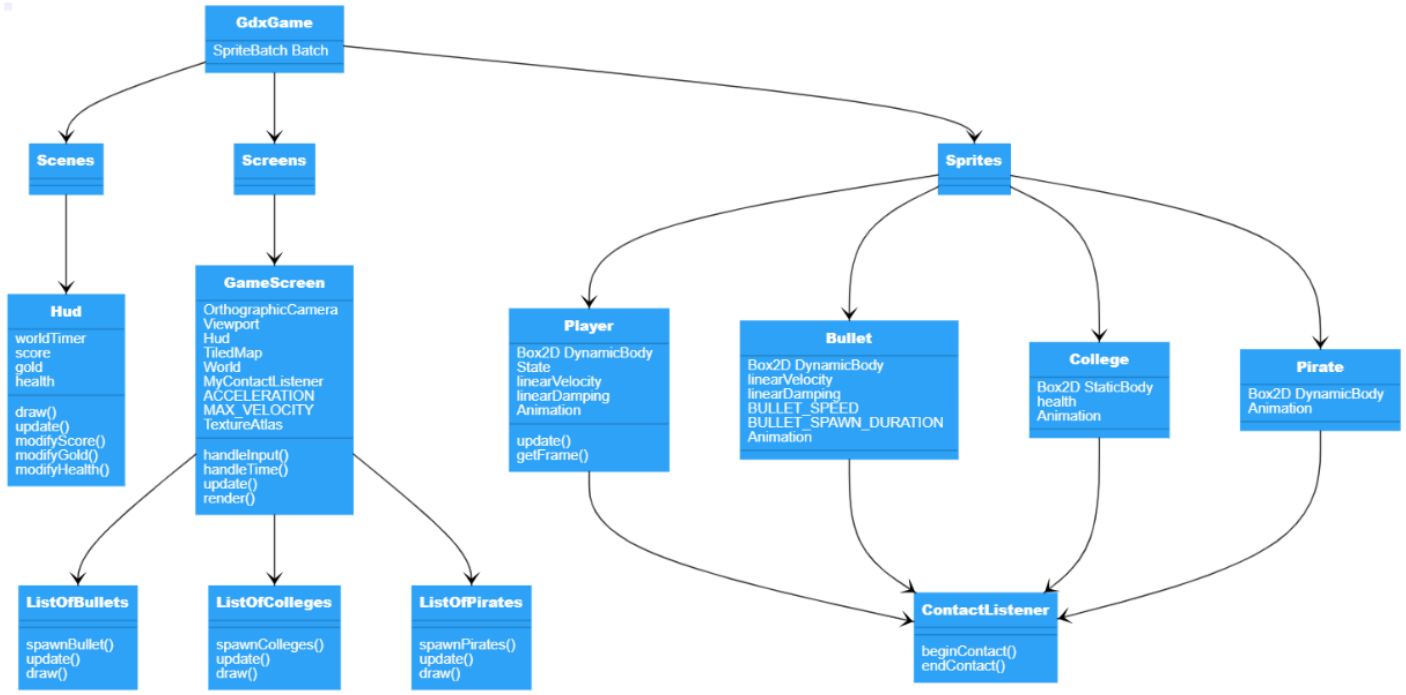
Viktor Atta-Darkua

Dandi Harmanto

Harsh Mohan

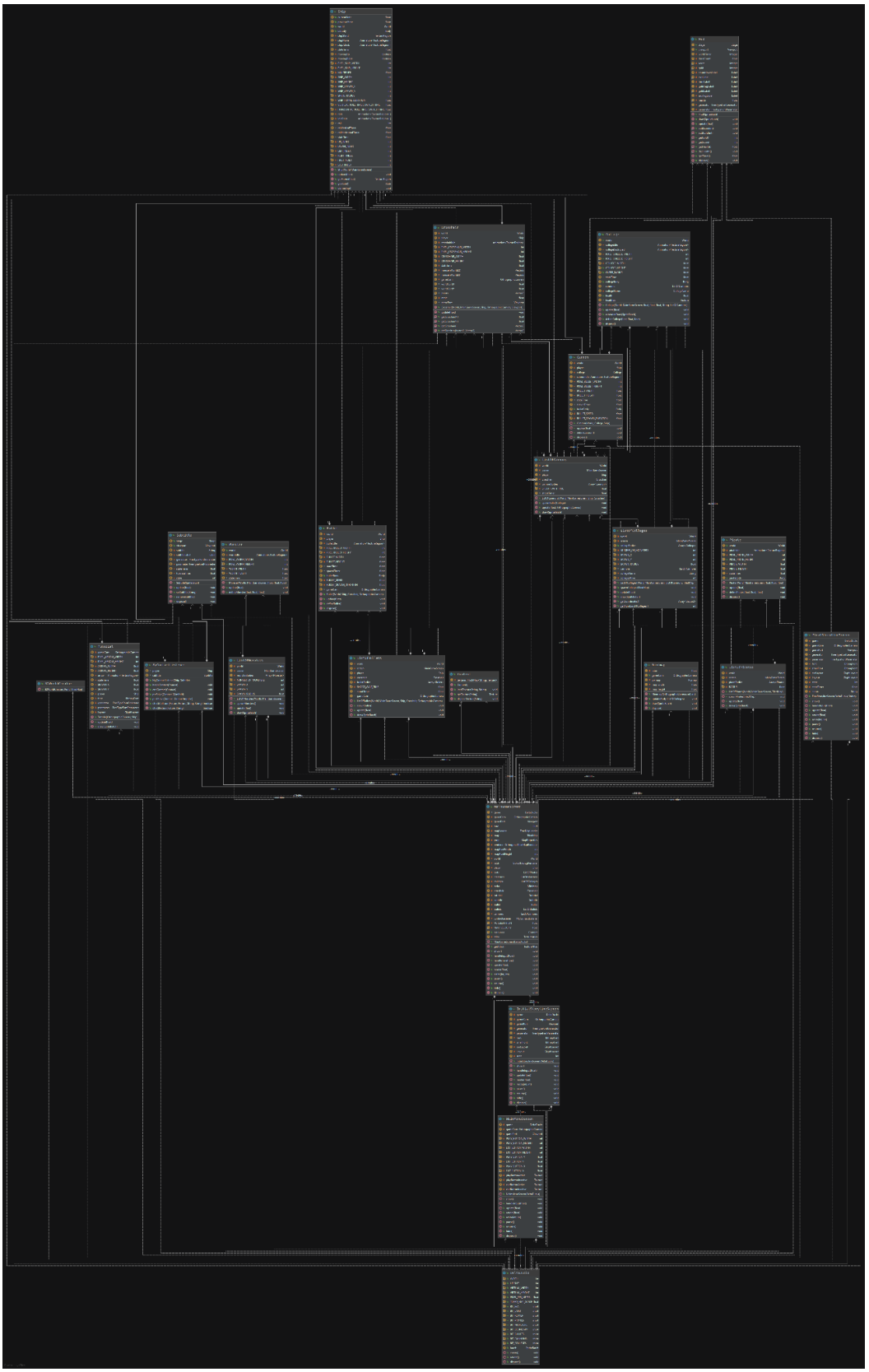
Danny Burrows

Abstract Architecture



Concrete Architecture

Link (As the UML may be too large to see clearly in the space we have): https://raw.githubusercontent.com/harshonyou/eng1-team8/main/deltaducks/diagram.png?token=GHSAT0AAAAAABQUSXKFOS4UU2FOAFOMGYYYYQDREDQ



Architecture Creation Language and Tools Used

We used UML (Unified Modelling Language) to create our architecture. UML is an object oriented modelling language used to plan and exhibit the design of a system and show how that system works.

In this case we used class diagrams to show the relationship between different entities, screens and pieces of code. This is usually done by splitting a rectangle vertically into 3 parts:

- The top part holding the name of the class / object.
- The middle part showing the variables / attributes of the class / object.
- The bottom part showing the functions in the class / object.

Lines can be drawn between different rectangles / classes to show the relationship between them, with arrow endings on a line showing an extension relationship and filled diamonds showing a composition relationship ([1] pg.54).

We used Gleek, found at https://www.gleek.io/, a free online software diagram building tool, to build our UML abstract architecture. The results can be found above.

For our Concrete Architecture, we later used IntelliJ IDEA and XML to.create a UML diagram to show the structure of the code we came up with once we had gotten around to finishing our game. The UML diagram clearly shows

- All the different screens, entities and classes in our code.
    - Their variables and function.
    - The relationships between them.

Systematic Justification For Architecture

Growth from Abstract to Concrete

Our abstract Architecture was used as a preliminary plan for how we would implement our program in the process of coding. The model is a bit prescriptive, based on our program requirements and planned approaches to implementing them.

The design includes an initial plan to the sprites, screens and scenes that would be required by our game:

- We first thought of what Sprites our game required
- We then thought about what screens our game required, predominantly our main game screen
    - Gave the groups of sprites that need to be spawned, e.g pirates, colleges, bullets… and linked them to it
- We then thought of the points and gold our game required, and decided to show them in the form of a Hud, which we defined as a scene
- We finally linked our scenes, screens and sprites to our overarching GdxGame class

Although this was used as an initial plan for our implementation, once we got around to coding it using libGDX a few things changed. The concrete architecture is much bigger than the abstract architecture, and also a bit different from it. This is because we had a lot of requirements our Abstract architecture could not fulfil that we needed to add in, some of these can be seen in the section below.

This, however, does not mean that the abstract architecture was useless. It contextualized most of our basic requirements in a form we could easily see, interpret and understand with our eyes, and helped us with our implementation by giving us a solid plan on what to start coding. The concrete architecture is necessary to provide an exhaustive representation of the final product, in doing so we have an unambiguous diagram of the result and this can more easily be applied to the requirements to assess the success of the project.

Requirements Fulfilled by Architecture's Constructs

Architecture Construct	Requirement ID	How Requirement is Fulfilled
MainGameScreen	UR_UNIQUE	MainGameScreen stores "player" as the ship belonging to and controlled by the player while all the AI controlled ships are stored in a list called "bots".
MainGameScreen	UR_ONE_SCREEN	MainGameScreen is designed to encompass the entire games function, displaying all elements in one place.
MainGameScreen	FR_SCALING	The resize method in MainGameScreen tracks the size of the screen and adjusts the resolution accordingly, ensuring the aspect ratio is preserved.
Subtitle	UR_TUTORIAL	MainGameScreen implements a tutorial

		function that is triggered upon starting a new game, this produces interactive overlays using the Subtitle class that the user can read to help them with getting familiar with controls and objectives.
MainGameScreen	FR_WASD	The handleInput method in MainGameScreen handles all inputs by the user, it specifically listens for the WASD keys and translates this to movement of the Player object.
Player, MyContactListener	FR_ATTACK	When attack events are registered by MainGameScreen the event is passed to the Player object. MyContactListener listens for bullets hitting AI ships or colleges and applies damage accordingly.
Pirates, MyContactListener	FR_AI_ATTACK	When the Pirate object fires a bullet, MyContactListener listens for bullets hitting the Player ship and applies damage if required.
Ship	FR_ATTACK_SUCCESS	When an AI Ship's health goes below zero that ship is removed from the map and the scores of the Player is updated accordingly.
HUD	FR_UI	The HUD object keeps track of all the UI elements on the screen, updating all the elements live and drawing them on the screen.
Pirates, ListOfPirates	FR_COMBAT	ListOfPirates is an object to store all the Pirates objects in the game. This list is iterated through when drawing all enemy ships to the screen.
MainMenuScreen	FR_NEW_GAME	Upon the user entering the game they are presented with a choice on the MainMenuScreen, if they click a new game then a new clear game is generated. At the end of the previous game the player is given the same choice via this screen.
Ship	FR_SHIP	If the player fires a bullet and that entity intersects with a ship that ship is removed from the map.
Pirates	FR_COLLEGE	The Pirates Object, one for each college, is instantiated at the beginning of the game, this then tracks the movement of the player in the world and fires on the play, engaging them in battle, if they come too close.
HUD, Player	FR_ATTACK_SUCCESS	The HUD object is responsible for displaying the plunder and points of the Player object, these are updated every frame and thus this ensures that the screen is updated in real time.

BiblioGraphy

[1] - [plantuml.com] [Updated Feb 2021] PlantUML Language Reference Guide (Version 1.2021.2)[Online] - Available https://plantuml.com/en/guide [Accessed 28th Jan 2022]