NTNU

Kunnskap for en bedre verden

Department of Computer Science

TDT4240 - Project Implementation Document

Haakon Svane:

MegaGolf

At the opposite end of minigolf

Group:
Group 29

Author:
Haakon Svane (haakohsv)

COTS:
iOS using Swift with SpriteKit, GamePlay Kit and Game Center

Primary quality attribute:
Modifiability

Secondary quality attribute:
Performance
Usability

April 28, 2021

Table of Contents

List of Figures

List of Tables

1 Introduction

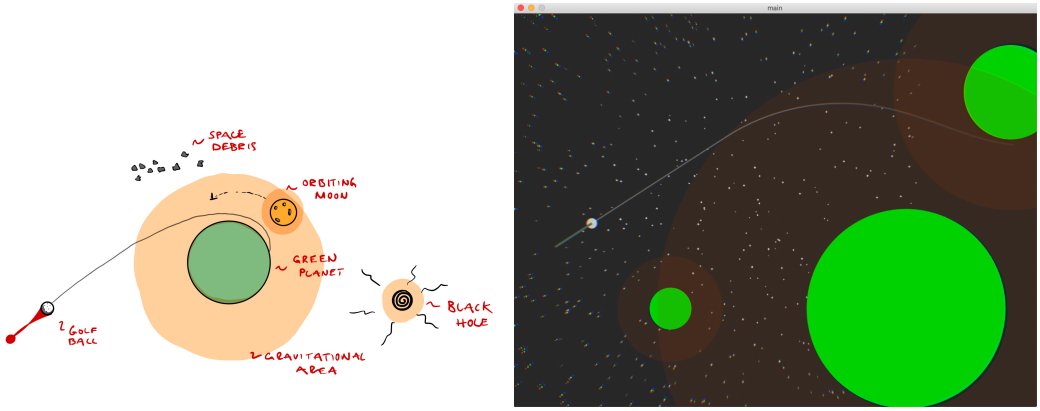
This section provides introductory information about the project, the game concept and the further structuring of this document.

1.1 Description of the project and this phase

This project is a semester project for TDT4240 - Software Architecture. The goal of the project is to design a mobile game (iOS or Android) with an online element to it. A software architecture is to be designed, documented and implemented. The purpose of this document is to record the implementation and realization of the project in light of architectural requirements and design decisions set before the implementation phase and to test these requirements.

1.2 Description of the game concept

MegaGolf is a golfing game set in space. It draws inspiration from both mini golf and regular golf. The aim of each level in the game is to make use of gravitational assist from planetary bodies to alter the trajectory of a golf ball, guiding it into a black hole which acts as a hole in regular golf. The game play is depicted in Figure 1a. A small proof of concept (POC) demo was made before project startup to evaluate the core mechanics of the game (gravity assisted trajectories). A screenshot of this demo is shown in Figure 1b.



(a) Illustration of the basic tokens of the game. The orange area around the celestial bodies show the gravitational area of effect.

(b) POC demo of the game to test the gravitational assisted trajectory and camera movement.

Figure 1: Game concept illustrated and demonstrated.

1.2.1 Game mechanics

Each level consists of one or several obstacles and one black hole. An obstacle may be a planet, an orbiting moon, a star or space debris. Planets come in the following categories: field planets, rough planets, water planets and bunker planets. The material that comprise the planets defines its category and their physical properties are similar to that one would see on a regular golf course. Space debris does not have any gravity so they act as deflecting obstacles rather than attracting obstacles. The player spawns some distance away from the goal (black hole) and must navigate to it by launching the golf ball using a drag and release command where the relative position of the finger at release time from the golf ball determines the velocity vector for the ball. The scoring of the game follows normal golfing conventions: each level has an associated par in which the difference between the player score and the level par determines the score awarded for the level. Lower is better.

1.2.2 Menus and modes

The main menu of the game presents a single player and a multiplayer mode. Single player gaming consist of unlockable levels that the player can unlock by completing previous ones. Levels are subgroups of solar systems such that a single solar system contains multiple levels. All levels in a solar system share a similar theme in terms of game play. The multiplayer portion of the game will consist of real-time player vs. player matching. Players will be able to invite friends or play vs random opponents around the world. An online game consists of playing through all levels in a solar system and the winner is the player with the best (lowest) score.

1.3 Structure of the document

This document describes the implementation phase of the project. Section 2 gives a comprehensive presentation of the specific implementation including all modules and models used in the implementation phase. Section 3 provides a manual on how to install, compile and run the project as well as a user manual for the game. Section 4 provides the results of testing the functional and quality requirements set for the system and Section 5 discusses the relationship between the implementation with respect to the planned architecture. Finally, Section 6 and 7 respectively present issues faced during the development of this project and its phase including some reflections as well as a list of contributions among the team members.

2 Design and Implementation Details

This section provides a detailed description of how the system was implemented and realized from the architectural requirements set for it including how the system interacts with the COTS products used with it. A complete class diagram for the project is available as an external resource (packaged with this document for project delivery) listing all the classes and models used in the implementation except Entities and Components. A components diagram reside in another file also attached with this document [1]. It is advised to use these as additional resources to this section when reading. The following subsections provide implementation details categorized by architectural granularity[2], lowest to highest.

2.1 MGPresenter

The `MGPresenter` class is a top-level controller for every aspect of the system (with the exception of global variables and user defaults.) From the perspective of Apple's provided framework for iOS applications, it is a subclass of `UIViewController` [Apple Inc., 2021g] which is responsible for managing the view hierarchy of an application. The MGPresenter is responsible for initializing the system, passing information between the managers and presenting other view controllers (GameCenter provides view controllers for matchmaking and Game Center account profiles). The MGPresenter owns three managers: `MGViewManager`, `MGSceneManager` and the `GameCenterManager` which provide interfaces for the MGPresenter to communicate with. The managers can only communicate with each other through the `MGManagerDelegate` which is a protocol that the MGPresenter conforms to.

[1] This diagram does however lack descriptions of entity component compositions. I had to exclude this because of time restrains, but the source code for each entity class clearly shows how they are composed. Not ideal, but so isn't hunger.

[2] Architectural granularity refers to the area of responsibility a module has in the system as a whole. If a module B is contained within module A, A is said to have lower granularity than B.

2.1.1 The Manager Delegate

In order to reduce coupling and to increase cohersion between the managers and the MGPresenter, the MGPresenter conforms to a protocol of methods that provide an interface for communication between the managers. For example, when the scene manager is notified about a value change in the game scene (e.g. golf ball is launched, the ball hit a goal etc.), the scene manager relays this information to the MGPresenter which is responsible for acting on it. The MGPresenter makes sure that this information is passed to the appropriate manager(s) or processes it itself.

2.1.2 MGViewManager

The MGViewManager is a container for functionality related to the `MGViews` of the system. An MGView is a system specific abstraction of a collection of GUI elements that fills the screen (see Section 2.2). Switching between the views is done through the privately owned `MGViewStateMachine` which contains `MGViewState` objects. These state objects contain view-specific logic like handling button presses, updating text, animating GUI element position based on entry / exit of the state etc. Each state owns the corresponding view and is responsible for initializing it. Since the initialization of the views happen at state initialization, the view manager can control which views are loaded into memory by managing which states are initialized or not. This is useful for managing memory footprint while switching between game play and menu navigation. This is realized through the MGViewManager methods `loadGameViews()` and `loadMenuViews()` where only the states needed to present the portion of the system is initialized.

2.1.3 MGSceneManager

The MGSceneManager is a container for functionality related to the `MGScenes` of the system. An MGScene is a system specific scene class inherited by Apple's `SKScene` [Apple Inc., 2021e] The MGSceneManager privately owns an instance of a `MGSceneStateMachine` where its states initialize the requested scenes. There are three types of MGScenes currently used by the manager. Below is a short description of each of them and how and when they are used.

MGMenuScene This scene holds all the instances used in the menu portion of the application. When a user navigates the menus, they are presented with a dynamically responding background scene showing the levels of the game. The initializer for this scene contains the logic for setting up the levels display as well as providing an interface for the MGPresenter to interact with it.

MGLoadingScreenScene The loading scene contains the content displayed upon loading which occurs when transitioning between an MGGameScene and a MGMenuScene. Since the actual loading of assets is done on a background queue, the loading scene can still be animated and is expected to perform relatively well.

MGGameScene This scene contains all the logic for the game. It contains a privately owned `MGGamePlayStateMachine` which contains `MGGamePlayState` instances. These states refer to the different states of the game play. For example, when the golf ball is at rest and is ready to be launched, the MGGamePlayStateMachine enters the MGAimingState. In this state, physics are disabled for the golf ball and the ball responds to user touch by showing an aiming indicator as well as a force indicator that provides information to the user on how the ball will be launched. If the player decides to launch the ball, the aiming state calculates the force vector for the ball and enters the `MGLaunchingState`. This state keeps track of the celestial objects that the ball has been in contact with and applies custom friction forces to the golf ball once it slides along an object. A new level in the game is created by defining a new subclass of the MGGameScene with naming convention `Level_[SYS_ID]_[LVL_ID]` where `SYS_ID` and `LVL_ID` respectively refer to the solar system and level identifier for the level.

2.1.4 GameCenterManager

The `GameCenterManager` contains all the functionality for integrating Game Center and its provided services to the system. When creating a GameCenter match, the matchmaking window can either be brought up modally using Apple's default design [Apple Inc., 2021b] or by designing a custom interface for it. Due to time restrains, the default View Controller provided by Apple was used. Some of the functionallity provided by the GameCenterManager is transmitting and recieving data to/from other players, notifying the MGPresenter that a player has been invited to a game, authenticating the local player and providing view controllers for matchmaking and Game Center login/sign-up. Transmitting and receiving data to and from other players is done by encoding and decoding a custom data model named `MGOnlineMessageModel`. A message using this model can be of type `LobbyData`, `ChatData`[3] or `GameData`. The GameCenterManager relays this message to the MGPresenter which ensures that the message is handled by the correct module based on the message type. For example, when the connected players are in a multiplayer lobby voting for matches, a `MGLobbyData` instance is encoded and transmitted to all other players once the local player presses the *READY* button. This specific instance of data is of type `MGLobbyData.isReady` and contains a boolean value indicating whether or not the player is ready as well as the player's currently voted map. During online game play, each local player unreliably transmits their current position and velocity to all other players. For any local player in the match, all other online opponent golf balls are of a special golf ball entity type named `GhostGolfBallEntity`. This is a stripped down version of the normal `GolfBallEntity` that a player interacts with, but lacks multiple components that make it interactive. This ghost ball enitiy does however simulate physics, meaning that since only position and velocity data is transmitted, if a transmission fails to deliver to a device, the local device will simulate the ghost ball to smooth out its movement.

2.2 MGView

An MGView is in essence a container for references to graphical user interface entities. A specific realization of a MGView, for example for the MainMenuViewState, is created by subclassing the MGView class and defining its appearance in the subclass initializer. The MGView subclasses SpriteKit's `SKNode` [Apple Inc., 2021d] so that the governing state may conveniently perform actions on the view as a whole if it desires. Figure 2 shows how one such view is realized for the main menu of the game and Figure 3 shows how the view is created in code.

When a view is added to a scene, it is done with a `isGUI` flag. Adding elements to the scene with this flag has the following consequences:

z-positioning The z-positioning of the view is set such that the whole view is placed on top of the rest of the scene content.

Ignoring the world node framebuffer All entities in the world node are sent to a private framebuffer for adding GSGL fragment shader effects to them[4]. The `isGUI` flag ensures that the elements ignore the effect of such shaders.

Camera positioning All elements with the flag has positioning relative to the camera position. This means that moving the camera around in a MGScene will not affect how the view is displayed.

Though the elements of each specific MGView needs to be added to the current MGScene in order to be displayed, they are completely decoupled from the MGScene instance from an architectural point of view. The managing state of each view (MGViewState) is responsible for communicating that the view is to be added to the current scene and the developer must specify the view size (usually the boundaries of the phone screen). In other words, the MGView is indifferent to any

[3]Though the model currently supports text chat, this has yet to be implemented into the game.

[4]There are a lot of bugs related to this, and a good work-around has not yet been found by the development team. This means that world node effects like black hole space warping and sun glare is currently not implemented in the game.

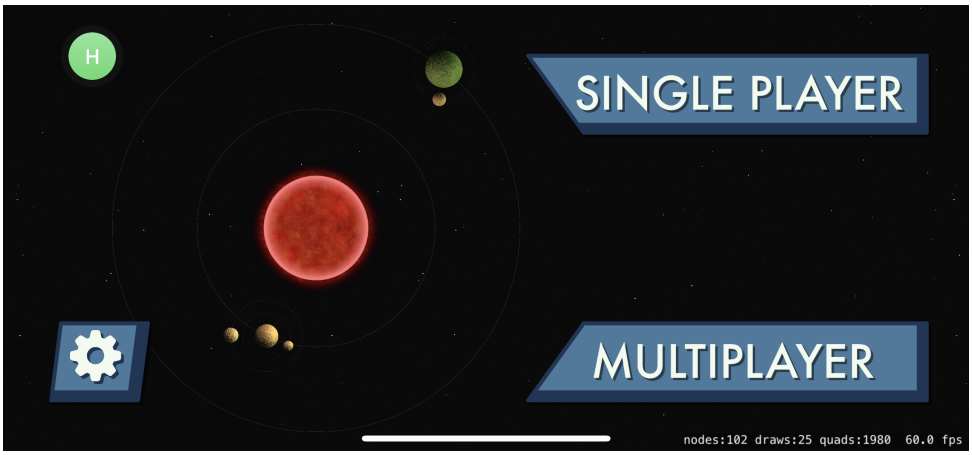


Figure 2: The `MainMenuView` displayed on top of the `MainMenuScene`. The GUI elements that are added from the view is the three blue buttons.

```
class MainMenuView : MGView{

    required init(managingState: MGViewState, size: CGSize){
        super.init(managingState: managingState, size: size)

        guard let manager = self.managingState! as? MainMenuState else{
            fatalError("The main menu view does not have a main menu state to manage it!")
        }

        let spButton = UIFactory.makeTopButton(buttonText: "SINGLE PLAYER", linkedClosure: manager.singlePlayerClick)
        spButton.getNode()?.position = CGPoint(x: size.width*0.25, y: size.height*0.3)

        let mpButton = UIFactory.makeBotButton(buttonText: "MULTIPLAYER", linkedClosure: manager.multiplayerClick)
        mpButton.getNode()?.position = CGPoint(x: size.width*0.25, y: -size.height*0.3)

        let settingsButton = UIFactory.makeSettingsButton(linkedClosure: manager.settingsClick)
        settingsButton.getNode()?.position = CGPoint(x: -size.width*0.4, y: -size.height*0.3)


        elements.append(spButton)
        elements.append(mpButton)
        elements.append(settingsButton)
    }
```

Figure 3: Snippet of code from the initializer of the `MainMenuView`. Each UI element is created through the UIFactory and the buttons reference a specific method in its governing state (the `MainMenuViewState`).

instances, inherited or not, of MGScenes. It is through its managing state and its communication with the MGPresenter that the view is added to the scene and modified.

2.3 Entities and Components

The system utilizes Entities and Components as a basis for all game tokens. This is not to be confused with Entity Component System (ECS) since there is no *system* present in the implementation of this architectural pattern. The reason for this (as explained in the Architectural Description Document) is that SpriteKit hides much of its rendering and physics pipeline from the developer. The specific implementation of this pattern follows the model presented in the Architectural Description Document. Because of the node hierarchies that are created for the components that require any type of SKNode, the implementation of the components document what other components are required. This is also enforced using run-time errors to ensure that the components behave as planned. Below are descriptions of some of the most important components used in the system.

NodeComponent The NodeComponent creates a top-level node of type `MGNode` which inherits from SpriteKit's provided SKNode. The implementation of the MGNode is done to make implementation of touch events for the node easier. The MGNode has an internal touch delegate property named `InternalTouchEventDelegate`. The delegate is at initialization

set to `nil`, but other components such as the TouchableComponent (which conforms to the delegate protocol) may process the touch events. The MGNode also contains a container for all child entities and a convenience method for adding other entities as a child to the node if they also possess a NodeComponent.

SpriteComponent The SpriteComponent creates an SKSpriteNode instance which it adds to the entities root-node (obtained from the NodeComponent). For textures that are contained within a `SKTextureAtlas` [Apple Inc., 2021f], the SKSpriteNode can be animated by setting its `isAnimated` flag to true in its initializer. If the texture is not contained within a SKTextureAtlas, normal maps can be provided with the regular texture to make the texture respond to lighting effects as if it was three-dimensional. This effect is seen on planets and the golf ball.

TouchableComponent Upon adding a TouchableComponent to an entity, the component sets the MGNode's touch delegate to itself. This makes the MGNode relay all touch events to the TouchableComponent and the response to these events can be overridden in the form of closures that are available from the TouchableComponent interface. For all UI buttons in the system, some of the behaviour is defined at entity initialization e.g. changing the button texture upon `touchDown`, playing sounds upon `touchUP` etc. For providing further functionallity to the buttons, a completion handler is called after the TouchComponent is done processing the internal entity specific logic for the touch event.

2.4 Property lists, the MGPlistParser and Global variables

In order to ensure that the primary quality attribute (modifiability) is prioritized, the system uses property lists (*.plist* files) to make modifying and adding functionality easier. Figure 4a and 4b show two examples of such lists.

Key	Type	Value
▼ Root	Dictionary	(11 items)
▼ GRAVITY_FIELD_BH	Dictionary	(2 items)
hex	String	#70D000
alpha	String	#20
▼ BALL_AIM_READY	Dictionary	(2 items)
hex	String	#457B9D
alpha	String	#FF
▼ ORBIT_LINE	Dictionary	(2 items)
hex	String	#FFFFFF
alpha	String	#30
▼ GAME_BOUNDS_LINE	Dictionary	(2 items)
hex	String	#E63946
alpha	String	#FF
▼ SLIDER_ON	Dictionary	(2 items)
hex	String	#E63946
alpha	String	#FF
▼ SLIDER_OFF	Dictionary	(2 items)
hex	String	#1D3557
alpha	String	#FF
▼ TEXT_RED	Dictionary	(2 items)
hex	String	#E63946
alpha	String	#FF
▼ TEXT_MAIN	Dictionary	(2 items)
hex	String	#F1FAEE
alpha	String	#FF
▼ TEXT_SHADOW	Dictionary	(2 items)
hex	String	#223D4E
alpha	String	#FF
▼ GRAVITY_FIELD	Dictionary	(2 items)
hex	String	#0070D0
alpha	String	#20

(a) Color properties list with some of its entries.

Key	Type	Value
▼ Root	Dictionary	(6 items)
▼ YellowStar	Dictionary	(4 items)
static friction	Number	1
kinetic friction	Number	1
restitution	Number	0
density	Number	120
▼ RedStar	Dictionary	(4 items)
static friction	Number	1
kinetic friction	Number	1
restitution	Number	0
density	Number	100
▼ MoonPlanet	Dictionary	(4 items)
static friction	Number	0,6
kinetic friction	Number	0,3
restitution	Number	0,7
density	Number	6
▼ BlackHole	Dictionary	(4 items)
static friction	Number	1
kinetic friction	Number	1
restitution	Number	0
density	Number	400
▼ FieldPlanet	Dictionary	(4 items)
static friction	Number	0,7
kinetic friction	Number	0,4
restitution	Number	0,3
density	Number	8
▼ BunkerPlanet	Dictionary	(4 items)
static friction	Number	1,8
kinetic friction	Number	1,4
restitution	Number	0,1
density	Number	7

(b) Planet properties list with its entries.

Figure 4: Snippets of two of the property lists used in the system.

Below is a list of descriptions for each of the property lists used in the system and their purpose.

Colors.plist Contains a dictionary of color values used throughout the system. Includes values for gravity field indicators, UI slider colors, text colors and sky color among other things.

DefaultProperties.plist Contains default parameters used throughout the game. Contains default player name, settings pane text, and default font for the application.

MGPlanetProperties.plist Contains physical data used in the initialization of the planets and stars in the game. Each planet has parameters for static friction, kinetic friction, restitution and density that determine how it interacts with other physics enabled entities.

MGLevelProperties.plist Contains the structuring of the solar systems and levels in the game. Each solar system has one star, a system identifier, a name and a list of levels orbiting around the star and each level has a name, an identifier and a specific design that is shown in the level select menu.

Parsing the property lists is done using a specialized subclass of a `MGPlistParser`. Some of the property lists values are set as global variables. These values are set by parsing the property lists at application startup. The exception to this is the *MGLevelProperties.plist*. This file contains information about how to build the solar systems for the menu scene. This ensures that new levels can be created fast and efficiently. The progress of creating a new level is summarized in the list below

1. Creating a new level file containing a class with naming convention `Level_[SYS_ID]_[LVL_ID]` as mentioned in Section 2.1.3. The class should inherit from `MGGameScene` and conform to `MGLevelSceneProtocol`.
2. Implementing the required initializer and calling the `MGGameScene` initializer.
3. Populating the level with entities using `GameObjectFactory` and appending the entities to the `MGScene` entities array as well as adding them to the scene node.
4. Adding the level to *MGLevelProperties.plist*. If a new level design (the visual representation of the level in the menu scene) is desired, this design is created in `LevelDesignFactory`. The level design is referred to in the properties list by string.

3 User manual

This section aims at providing information on how to install, compile, run and play the game.

3.1 Installing and/or compiling the game

The application runs on all iPhone devices with operating system version 14.1 or higher. It is tested on an iPhone Xs operating system version number 14.4.X. It is adviced to run the project on similar hardware and software as no guarantee can be made for any differing versions. Installing and compiling the source code on another system is not advised as it *may* prove cumbersome. Section 3.1.1 provides information on things to consider if this is to be done while Section 3.1.2 provides information on how to install the application using Test Flight.

3.1.1 Compiling and running the source code

With the project package delivered together with this document, the full application bundle is made available. Opening this bundle requires Xcode and a macOS operating system. This bundle is tested on Xcode 12.4 with macOS 10.15.7 and if it is to be imported to another system, it is advised that the version numbering match these. In the root directory of the project bundle lies a *info.plist* file. Depending on the specific details regarding the developer certificate of the target user system, the properties of this list may have to be changed to fit the user credentials. This also applies for the code signing settings in the main bundle under target *MegaGolf*. If all else fails, the source code files could always be manually imported to a new blank Xcode project. In this case, make sure that *Main Interface* and *Launch Screen File* in *Bundle Settings* $\longrightarrow$ Target: *MegaGolf* $\longrightarrow$ *General* are both set to *Main*. You may also be promted to manually set the target iOS version to 14.1 because of framework dependencies.

3.1.2 Installing via Test Flight

In addition to the full source code for the project, the application will be made available for testing on Test Flight with a public invite code. This code will be provided once the final beta build is approved for beta testing by Apple. Beta app approval usually won't take long, but there is no guarantee that this code will be available at project deadline. This code is therefore to be presented once available in the *README.md* in the provided git repository. Installing the game using Test Flight requires the user to first install Test Flight from the App Store. Entering the provided public beta code is done on the main page of Test Flight by pressing *Redeem* and entering the code. Once this is done, the user will be taken to a beta page for the application where it will be possible to install the game to the local iPhone.

3.2 How to play the game

This section aims at providing a user guide on how to play the game from application start to finish. The subsections are divided into menu navigation and game play description.

3.2.1 The menus

Main menu
Upon application launch, you are presented with the main menu of the game as shown in Figure 5. From this menu, you may choose between playing single player mode, multiplayer mode or change system settings.

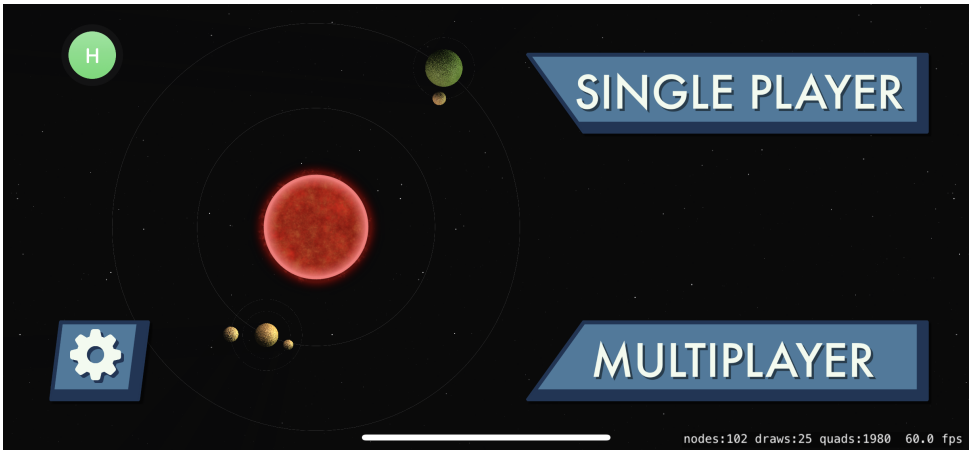


Figure 5: Main menu screen of the game.

Settings pane
The settings pane is shown in Figure 6. This pane is accessible from multiple parts of the application. Inside the settings pane, you may choose to set the volume for both the music and sound effects individually. The preferences will be saved for other sessions. The question mark button presents an *About* menu, but this is currently not implemented so pressing the button will have no effect.

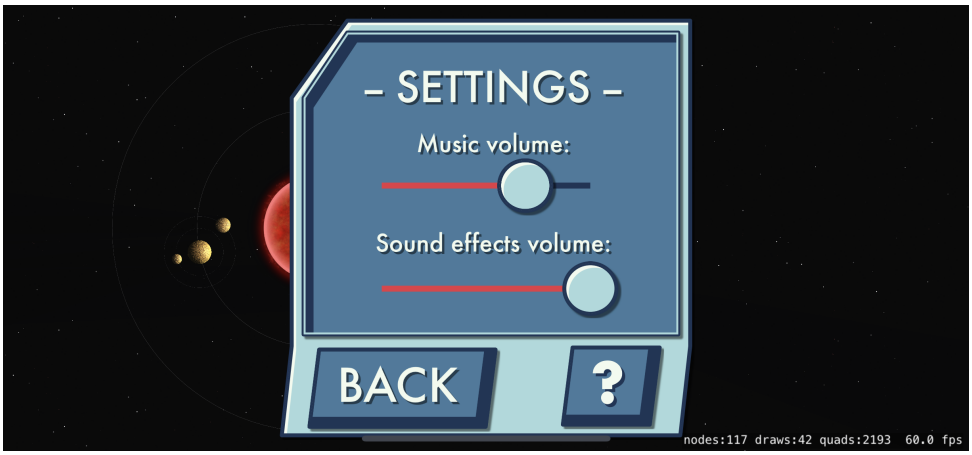


Figure 6: Settings pane in the game.

Single player
The single player button takes you to a level selection. This menu is shown in Figure 7. Upon entry, you can browse the available solar systems by pressing the left and right navigation buttons.

Figure 7: Level select menu upon entry.

Upon pressing the *SELECT* button, you are presented with the levels available in the solar system with their names as shown in Figure 8. You may browse the available levels by navigating with the left and right arrow buttons just like with the solar systems. In the current version of the game, you may play all available levels. For later versions, you will have to play the first level in a solar system to unlock the next. The design of the level gives you a hint of what to expect from that specific level. You may at any time go back to either the solar system selection or the main menu by pressing the back button in the lower left corner. Playing the level is done by pressing the *PLAY* button. This will take you to a loading screen before the game starts

Figure 8: Level selection of a specific map.

Multiplayer
Selecting *MULTIPLAYER* from the main menu takes you to a multiplayer screen. By pressing the *FIND* button in the lower left corner, you will be presented with a Game Center matchmaker as shown in Figure 9. You may set the number of players to play with (2 - 4) and invite players that are nearby or in your friend/contact list. If the number of invited players are less than the maximum number of players you set in this menu, the matchmaker will automatically find other players currently searching for a game. You may exit the lobby at any time. Pressing the *EXIT* button will display an alert requiring you to confirm. The other players will be notified about a player leaving the lobby.

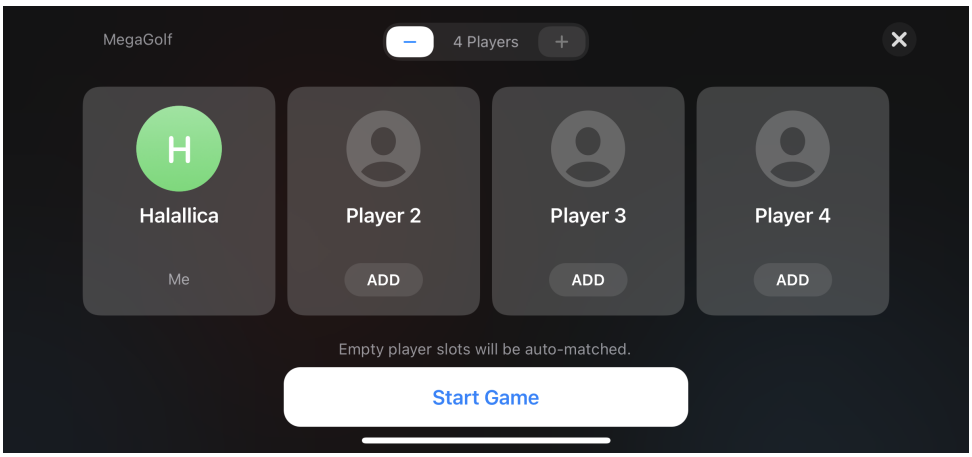


Figure 9: Matchmaking is done through the Game Center controller.

When all players have connected to the match, the game will display a multiplayer lobby as shown in Figure 10. In the lobby, you will see all the connected players, their ready status and options to vote for the solar system that the players should play through. When all players are ready, a countdown appears before the game starts. The lobby also shows the system that was chosen based on the votes from the players. The system is chosen randomly following the distribution of votes, meaning that if 2 players select system 1, 1 player selects system 2 and 1 player selects system 3, the respective chances of playing the systems are 50%, 25%, and 25%.

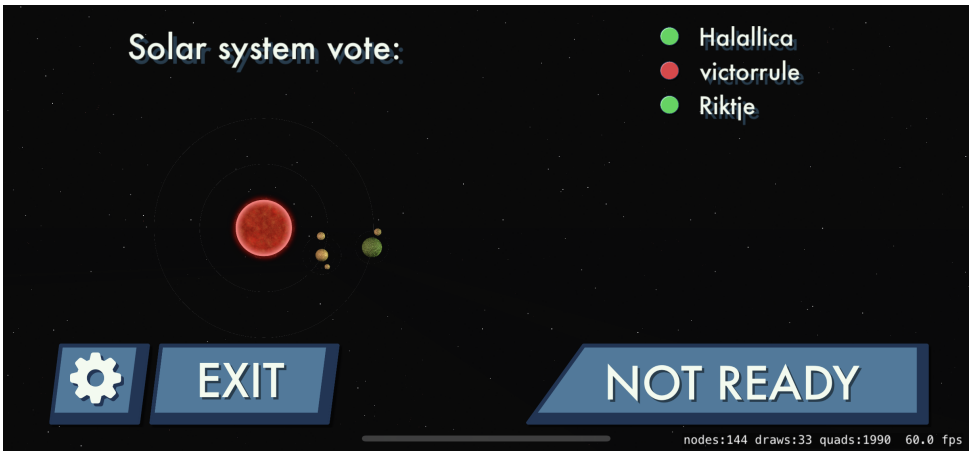


Figure 10: The multiplayer lobby.

3.2.2 The game play

The aim of the game
The aim of the game is to launch a golf ball into a black hole that is located somewhere on the level. A view of the game play is shown in Figure 11. When the ball is ready to be launched, a blue circle appears around it. Touching and dragging the finger from the ball displays a aiming indicator as well as a force indicator. The greater the distance between the dragging finger and the ball, the more powerful the launch will be. This is indicated by the color of the force indicator.

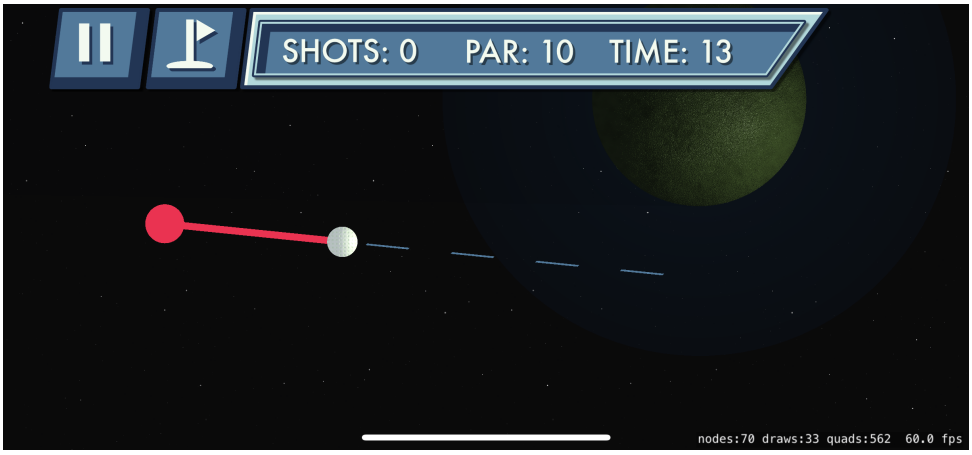


Figure 11: A view of the game play as the ball is being aimed for launch.

Planets and other celestial objects attract the golf ball if the ball is close enough to it. The gravitational area around each celestial object is indicated by a blue circle around it. Outside this region, the ball is freely floating in space.

Throughout the game play session
The top portion of the screen contains three elements. The first (top left) is the pause button. Pressing this button pauses the game and takes you to a game paused screen where you may go into the settings pane or exit the game. Returning to the game play is done by pressing the back button.

The player may wish to see the location of the goal at any point in the game. This can be done by pressing the flag pole button next to the pause button. This smoothly pans the camera towards the goal, letting the you see where the goal is and what stands between you and it.

The Planets
The planets act as either hazards or intermediate stops for the golf ball. The physical properties of a planet determines the behaviour of impact between it and the golf ball. For example, a Field Planet is a smooth grass planet. This means that the ball is expected to roll a little when landing on it. The Bunker Planet is however not very smooth. Though the ball is by no means lost when it lands on it, launching from the planet is harder, and the ball is not expected to roll much before coming to a rest. Planets such as the Water Planet is considered a hazard and should be avoided. Landing on this planet causes the ball to dissapear. This resets the position of the ball to the last safe resting point.

Deep Space
The level boundaries are marked with a red dashed line that all level entities are contained within. If a golf ball exits this area, a text pop-up appears. Entering deep space is considered a hazard, so the position of the ball will be reset to its last safe resting spot.

Scoring
The score given to the player follows a modified version of that you would except from regular golf. Each level has a time limit (`maxTime`) and a time elapsed (`timeElapsed`) from the start of each level. In addition to these parameters, a player is given a score based on the difference between shots made and the level par (`shotScore`). Upon entering the black hole, the player is presented with this score and its associated name. Table 1 lists all the namings for the different shot scores.

Shot score	**Name**
-4	Condor
-3	Albatross
-2	Eagle
-1	Birdie
0	Par
0 (first shot)	Black Hole in One
$+1$	Bogey
$+2$	Double Bogey
$+3$	Triple Bogey

Table 1: Scoring of the game with their associated names. Values below or over the shot score have no associated names, but still count.

The final score of each level is then determined based on the following equation

$$\texttt{totalScore} = 100\left(1 - \frac{\texttt{timeElapsed}}{\texttt{maxTime}}\right)\left(1 + \tanh\left(-\frac{1}{4}\texttt{shotScore}\right)\right) \tag{1}$$

After the game
After the goal is hit, the player is notified about the score achieved. The player is then taken to a results screen. From this screen, the player may choose to either exit the game, or play the next map in the solar system (if any). The feature of playing the next map is however currently not supported. This means that after both online and single player games, the player must exit to the main menu after each level. For online matches, this disconnects the player from the others.

4 Test Reports

This section presents the result from testing the functional and quality requirements.

4.1 Functional requirements test reports

The functional requirements set for the system where tested for success. This subsection lists all the functional requirements for the project and presents the evaluation of them.

ID: FR1	System should respond to touch control.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	1 minute
Evaluation:	Success
Comment:	The system responds to user touch where necessary.

Table 2: Test results from functional requirement FR1.

ID: FR2	Launch velocity of ball is controlled by drag and release.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	2 minutes
Evaluation:	Success
Comment:	Ball is launched at different velocities based on the relative distance between it and the touch down position as expected.

Table 3: Test results from functional requirement FR2.

ID: FR3	All game modes are presented on the main menu.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	1 minute
Evaluation:	Success
Comment:	Both single player and multiplayer mode are presented on the main menu of the game.

Table 4: Test results from functional requirement FR3.

ID: FR4	The game should support single player mode.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	1 minute
Evaluation:	Success
Comment:	The single player mode is present on the main menu screen.

Table 5: Test results from functional requirement FR4.

ID: FR4.1	The single player mode should display a scoreboard / par info pane.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	2 minutes
Evaluation:	Success
Comment:	The par info pane is displayed in the top middle portion of the screen during game play. Note that at the current time, the resulting score is in no way presented to the player upon level completion.

Table 6: Test results from functional requirement FR4.1.

ID: FR5	The game should support real time online multiplayer.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	4 minutes
Evaluation:	Success
Comment:	The game connects up to 4 players for an online real time session. Data is transmitted and handled on all devices. Note that there is no real objective to the multiplayer game at the current time.

Table 7: Test results from functional requirement FR5.

ID: FR5.1	The multiplayer should utilize Game Center services.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	2 minutes
Evaluation:	Success
Comment:	Game Center services are used for all online aspects of the game, be it matchmaking or data transmission between devices.

Table 8: Test results from functional requirement FR5.1.

ID: FR5.2	A player should be able to invite friends for the multiplayer session.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	2 minutes
Evaluation:	Success
Comment:	As long as a player is authenticated to Game Center, inviting friends from both the contacts list and the Game Center friends lists works.

Table 9: Test results from functional requirement FR5.2.

ID: FR5.3	The multiplayer session should auto-fill opponents.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	3 minutes
Evaluation:	Success
Comment:	If a player desires more opponents than has been invited, Game Center will match-make other players if available.

Table 10: Test results from functional requirement FR5.3.

ID: FR5.4	The multiplayer session should provide a text / symbol chat.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	N/A
Evaluation:	Not implemented
Comment:	Not implemented

Table 11: Test results from functional requirement FR5.4.

ID: FR5.5	The multiplayer session should provide a scoreboard / par info pane.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	5 minutes
Evaluation:	Success
Comment:	Though there are no ways to see the scores of the other players, a local par info pane is displayed in the top middle portion of the screen during game play. This scoreboard only reflects the local player status. The requirement is deemed a success since it is technically is satisfied.

Table 12: Test results from functional requirement FR5.5.

ID: FR6	The game should support a map-maker mode.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	N/A
Evaluation:	Not implemented
Comment:	Not implemented

Table 13: Test results from functional requirement FR6.

ID: FR7	Physical entities should respond to collisions and or gravity.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	3 minutes
Evaluation:	Success
Comment:	All entities with a PhysicsComponent detects collisions and all entities with GravityComponent exerts a gravitational force on others.

Table 14: Test results from functional requirement FR7.

ID: FR8	The game should support dynamic sound.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	N/A
Evaluation:	Not implemented
Comment:	The audio engine does not support support dynamic sounds at the moment. The soundtracks have been designed for this.

Table 15: Test results from functional requirement FR8.

ID: FR9	The game should have a settings pane.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	2 minutes
Evaluation:	Success
Comment:	The settings pane is available from multiple states of the application.

Table 16: Test results from functional requirement FR9.

ID: FR9.1	The settings pane should never be more than one view away from the user.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	10 minutes
Evaluation:	Failure
Comment:	While this is technically a success (from a code perspective), when choosing a level in a solar system, the distance is a minimum of 2 to the next settings button (main menu or game play paused view).

Table 17: Test results from functional requirement FR9.1.

ID: FR9.2	The settings pane has individual sliders for sound effects and music.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	2 minutes
Evaluation:	Success
Comment:	The settings pane provides two sliders for both sound effects (UI click sounds and ball launch sound) and music (in the menus or in a level). The effects are instantly handled while sliding.

Table 18: Test results from functional requirement FR9.2.

ID: FR10	The game should support achievements through Game Center.
Executor:	Haakon Svane
Date:	2021-04-23
Time used:	N/A
Evaluation:	Not implemented
Comment:	Not implemented. Once set, achievements can not be changed. It was therefore decided to wait with this until game development completion.

Table 19: Test results from functional requirement FR10.

4.2 Quality Requirements test reports

The quality requirements set for the system where evaluated based on the expected response measure and the observed response measure. This subsection lists all the functional requirements for the project and presents the evaluation of them. In the tables, *Expected* and Observed refers to the expected and the observed response measures of the requirement.

ID: M1	Adding / reworking planets.
Executor:	Haakon Svane
Date:	2021-04-24
Environment:	Design time
Stimuli:	Developer wants to implement a new planet
Expected:	Within 30 minutes
Observed:	21 minutes
Evaluation:	Success
Comment:	For this test, a water planet was designed, implemented and tested in both single player and multiplayer. Designing the texture of the planet is not taken into account for the observed measure, but creating its particle emitter is.

Table 20: Test results from modifiability requirement M1.

ID: M2	Adding / modifying levels.
Executor:	Haakon Svane
Date:	2021-04-24
Environment:	Design time
Stimuli:	Developer wants to add a new level to the game
Expected:	Within 2 hours
Observed:	55 minutes
Evaluation:	Success
Comment:	For this test. A new level was designed and added to an existing solar system. The level design (visual representation of the level on the level select screen) was also created. The results where tested in both single player and multiplayer.

Table 21: Test results from modifiability requirement M2.

ID: M3	Adding / changing textures.
Executor:	Haakon Svane
Date:	2021-04-23
Environment:	Design time
Stimuli:	Developer wants to add / change textures
Expected:	Within 30 minutes
Observed:	6 minutes
Evaluation:	Success
Comment:	For this test, the background star sky tile map images where changed such that the resolution of each image was doubled. The TileMapComponent handles this change dynamically and the observed time reflects the time from the texture was imported to the application bundle to the time that the change is confirmed to be working as intended in single player mode.

Table 22: Test results from modifiability requirement M3.

ID: M4	Modifying component behavior.
Executor:	Haakon Svane
Date:	2021-04-23
Environment:	Design time
Stimuli:	Developer wants to change component behaviour.
Expected:	Within 30 minutes
Observed:	16 minutes
Evaluation:	Success
Comment:	For this test, the gravitational fields where modified to work more consistently over different planet sizes. The template planet entity module was affected. The observed time reflects the time from opening up the IDE to verifying the change over a range of planet sizes.

Table 23: Test results from modifiability requirement M4.

ID: P1	System response time of user interface touch event
Executor:	Haakon Svane
Date:	2021-04-23
Environment:	Run time
Stimuli:	The user presses a UI button
Expected:	0.1 seconds
Observed:	$0,07357$ seconds
Evaluation:	TEXT
Comment:	With $n = 27$, the average time from button press recognition to state transition was $0,07357$ seconds with a standard deviation of $0,01905$ seconds. Note that the time measured is the time between touch down recognition to touch up response. This means that holding down the button for longer would affect the measurement. The actual system response happens faster since the a button down press is indicated by a change of button texture so the results are well within the frames of the requirement.

Table 24: Test results from performance requirement P1.

ID: P2	Periodic system update throughput.
Executor:	Haakon Svane
Date:	2021-04-24
Environment:	Run time
Stimuli:	Periodic system update
Expected:	60 FPS with 80% stability
Observed:	$59,9700$ FPS with $99,3172\%$ stability (menus) $59,9699$ FPS with $99,6980\%$ stability (game play)
Evaluation:	Success
Comment:	The data used in the test consists of 1612 frames of navigating the menus normally (all states visited) and 2319 frames of game play with normal playing (all states visited).

Table 25: Test results from performance requirement P2.

ID: U1	Interactions with non-interactibles.
Executor:	Haakon Svane
Date:	2021-04-24
Environment:	Normal operations
Stimuli:	Developer wants to implement a new planet
Expected:	Less than 5 interactions with non-interactibles for the first 30 minutes of usage.
Observed:	4
Evaluation:	Success
Comment:	For this test, a user with no experience from the game was used. The user has much experience with iOS games. The user tried to click outside of the settings pane and pause pane to make it disappear, drag the game scene to scroll the map area and tried to interact with the solar systems on the level select. Consecutive repeated actions are not counted.

Table 26: Test results from usability requirement U1.

ID: U2	Basic controls information.
Executor:	Haakon Svane
Date:	2021-04-24
Environment:	Normal operations
Stimuli:	User starts the game for the first time
Expected:	User does not forget the basic controls for the same game session
Observed:	Not tested
Evaluation:	Not tested
Comment:	Not tested

Table 27: Test results from quality requirement U2.

ID: U3	Asset loading presents a loading screen.
Executor:	Haakon Svane
Date:	2021-04-24
Environment:	Normal operations
Stimuli:	User initiates state transition
Expected:	Asset loading that will lead to frame stutter is hidden with a loading screen
Observed:	Some stuttering artefacts due to repeated texture loading
Evaluation:	Failure
Comment:	While the loading screen hides most of the scene initialization, SpriteKit's dynamic asset loading causes frame stutters when unused scene textures are introduced for the first time in a scene. This is expected to be fixed once a proper texture manager is developed at a later time.

Table 28: Test results from quality requirement U3.

5 Architectural Consistency

In general, much of the implementation follows the architecture that was designed. The abstract framework that was developed in the architectural design process is mainly the cause for this. Without any experience of the COTS frameworks that was used in the implementation, designing a rigid and detailed architecture and planning design patterns around them are hard. The following subsections presents areas of the system that differ from the designed system architecture.

5.1 Architectural patterns

This section focuses on the architectural patterns that where introduced in the architectural design phase and how they relate to the specific implementation of them.

5.1.1 Entities and Components

Coupling between modules
The specific architecture of the Entities and Components system is in tact with the architectural design. There are however issues that arise from the specific implementation around the node hierarchy that was implemented. Since this solution introduces dependencies between components (mainly that many of the components are dependent on a NodeComponent), coupling between them is increased. This was however alleviated a little by requesting the NodeComponent before communicating with it.

Area of effect
The degree of the usage of the Entities and Components pattern can vary in a system. One implementation might for example chose to strictly follow this pattern for all modules. This requires a highly abstracted implementation of the Entity and Component pattern and since the core functionality of this is provided by Apple's GamePlayKit [Apple Inc., 2021a], this proved a bigger challenge than expected. Entities and component where therefore only used for ingame tokens. Lower level functionality such as property list parsing and networking was omitted from this pattern. This could have been discussed and noticed in early phases of the project for example in the ATAM session.

Breaking the pattern
Entities and Components is a pattern that favours composition instead of inheritance. To make some entity creations quicker, abstractions for some of the entities was created in the form of template classes that specific entities inherit from. An example of this is the `MGPlanetEntity` that many of the specific planet entities inherit from. This implementation was done to increase the modifiability of the system.

5.1.2 Model-view-presenter

Boilerplate code
The model-view-presenter pattern was implemented in the top-level of the system. The specific implementation of this is highly dependent on the implementation of the state pattern for the view states. While the core idea of the MVP pattern is still present, the specific implementation may somewhat reduce the modifiability of the system since some boilerplate code is required when creating new views. This issue is hard to resolve early on in the design phase since it is very implementation dependant.

Scenes
Deciding where the scenes should be put inside the MVP pattern is a little tricky. They are both responsible for initializing entities, updating them, presenting them to the screen and driving internal game state changes (game play states). The addition of the abstracted UI views in the system (which still require the scenes for adding elements) sort of creates a triangle of dependancies between the scenes, the presenter and the views. To mitigate some of these dependancies, all communications between a scene and a MGView must happen through the ManagerDelegate that the MGPresenter conforms to.

5.2 Design patterns

This section focuses on the design patterns that where introduced in the architectural design phase and how they relate to the specific implementation of them.

5.2.1 Factory pattern

Factories as singletons
While the factory pattern is extensively used in the system, a implementation decision was to make the factories static (singleton). This was done to reduce memory footprint and increase the performance of the initialization processes.

Performance issues due to lack of proper resource pooling
In the current implementation of the factories, only some texture management is done by the use of static pointers to the textures in memory (resource pooling). This was done because of a bug in SpriteKit that caused normal maps to be applied incorrectly if either the textures or the normal maps where contained within a SpriteKit SKTextureAtlas. The affected textures therefore has to be loaded as image files and stored as separate instances. A texture manager is in the process of being developed, but due to time restrains, this was never properly merged with the

component system and so is not being used by the system. This creates an inconsistent use resource management by the SpriteComponent.

6 Issues and project reflections

This section presents issues that were faced during the development of this phase as well as some reflections on the project as a whole.

6.1 Reflections

Working on a project this size is exiting. Due to the circumstances of the project, I was presented with a unique opportunity of working alone instead of in groups of 4-6 people as this project usually requires. This has caused issues from both a personal and and an academical point of view, but I wouldn't change anything around these circumstances if I were to do this again. I started this semester knowing nothing about iOS and Swift development, and my past experiences with game development was next to nothing. Ignoring all issues, here are some considerations worth mentioning.

6.1.1 Riding Solo

Working alone can really go both ways. Having no partners to bounce ideas off of can cause a vacuum chamber which can and will cause issues during the development of the system. The upside to working alone however, is that both design and implementation of the system can be streamline easier. There is no work to be done in parallel and design and implementation philosophy is much easier to maintain.

6.1.2 Apple's framework for game/application development

In contrary to my prior belief in Apple's commitment to quality, their frameworks can be really frustrating to work with. While they provide great functionality which is easy to adapt to, SpriteKit in particular has many issues that require dwelling deep into forums to find work-arounds to. Then again, I think software engineering is one of those professions where it is mandatory for each engineer to first spend a great portion of his time complaining about other people's work before actually doing something valuable.

Apple has recently started porting their graphics framework from OpenGL to Metal (their proprietary graphics API). I once looked forward to reaping the fruits of having a game run on a chip with a graphics API specially tailored for it, but boy was I wrong. My only real interaction with Metal (note: through SpriteKit) has been in the bundle settings where I have disabled all Metal warnings and API checks since my own console outputs are impossible to find in an ocean of useless warnings.

6.1.3 An Ode to my Girlfriend

This last section is a symbolic gesture to my girlfriend who has had to spend many days and nights alone. I am grateful for the many lunches and dinners I have been made and to the many hours of listening to me ranting about algorithms, physics and mathematics.

6.2 Issues

Table 29 lists issues that were faced during the development of this phase (implementation and testing). Table 30 lists known bugs and unwanted behaviour currently in the application.

Date	Issue	Description
2021-03-07	Normal maps	Normal maps are incorrectly applied when either the object texture or its normal map reside in an SKTextureAtlas. This bug has been reported to Apple.
2021-03-14	Metal API	Mutliple Metal API warnings are issued when using custom GSGL fragment shaders on SKShapeNodes. This bug has been reported to Apple.
2021-03-20	Private framebuffer	Setting the `shouldEnableEffects` flag in the world node of type `SKEffectNode` [Apple Inc., 2021c] causes unwanted cropping effects to occur. World node fragment shaders can not be implemented because of this. This bug has been reported to Apple.

Table 29: Issues during development of this phase.

Problem	Description
Loading stutter	Upon first sight of a texture, the game stutters when performing asset loading operations.
Funky button text	Pressing a labeled button with multiple fingers makes the text move away from the button center.
Bad states	Rapidly pressing multiple buttons at the same time can cause the game to soft lock into some states.
Water hazard	Upon touching a water planet, the ball freezes in place instead of following the planet as it fades.
Particle emitters	Particle emitters behave weird at start/stop.
Sound issues	Scene transitions cause sound popping.

Table 30: Known bugs and unwanted behaviour.

7 Changes and contributions

Table 31 lists all the changes that was made to this document over time with dates and descriptions. Table 32 lists the individual contributions made to the project at this phase with estimated hours of work.

Date	Change	Description
2021-04-26	First release	Initial revision of the document.

Table 31: Changes made to this document over time.

Name	Description	∼Number of hours
Haakon Svane	Written the implementation documentation. Implemented the system and tested it.	360

Table 32: Individual contribution of team member(s) for this document.

Bibliography

Apple Inc. Framework: Gameplaykit, 2021a. https://developer.apple.com/documentation/gameplaykit [Accessed: 2021-04-24].

Apple Inc. Class: Gkmatchmakerviewcontroller, 2021b. https://developer.apple.com/documentation/gamekit/gkmatchmakerviewcontroller [Accessed: 2021-04-22].

Apple Inc. Class: Skeffectnode, 2021c. https://developer.apple.com/documentation/spritekit/skeffectnode [Accessed: 2021-04-24].

Apple Inc. Class: Sknode, 2021d. https://developer.apple.com/documentation/spritekit/sknode/ [Accessed: 2021-04-15].

Apple Inc. Class: Skscene, 2021e. https://developer.apple.com/documentation/spritekit/skscene/ [Accessed: 2021-04-16].

Apple Inc. Class: Sktextureatlas, 2021f. https://developer.apple.com/documentation/spritekit/sktextureatlas [Accessed: 2021-04-22].

Apple Inc. Class: Uiviewcontroller, 2021g. https://developer.apple.com/documentation/uikit/uiviewcontroller [Accessed: 2021-04-21].