|  |
| --- |
| Technical Design Document |
| Virtual Baby Seal Pet |

|  |
| --- |
| Jaclyn Moorhouse  m011845m |

# Glossary

**Game Object** - A Unity object that represents characters, UI or items

**Prefab** – A reusable Game Object template in Unity

**Component** – A modular script or feature attached to a game Object

**Scriptable Object** – A Unity asset used to store data independently of Game Objects

**Scene** – A level in Unity

**Asset** – any image, sound, script or object in the game

**Seal** – The player’s virtual pet

**Stat decay** – the gradual reduction of a pet’s values (e.g. hunger or hygiene) over real time

**Reward Wheel** - A feature that gives player prizes every 12 hours

**XP** – Experience points gained through actions used for levelling up

**Hygiene** – A stat representing how clean the seal is

**Energy** – A stat representing how rested the seal is

**Seal Name** – A customisable string entered by the player to name the pet

**Singleton** – A design pattern that restricts a class to one instance

**State Machine** – A system where an object can be in one of many states, each which trigger specific behaviours

**Observer pattern** – A design pattern where objects listen and respond to changes in other objects

**Flyweight pattern** – A pattern that minimizes memory use by sharing as much data as possible

**Coroutine** - A unity Feature for running actions over time

**On Trigger Enter** – A Unity method that detects when a collider marked as trigger is entered

**Input Field** – A UI element allowing the player to type text

**Audio Clip** – A sound file used in Unity

**Audio Source** – The Unity component that plays sound

**SFX** – Sound effects or short audio clips

**Touch Input** – Interaction based on screen taps or swipes

**Drag and drop** – A gesture allowing items to be moved and placed

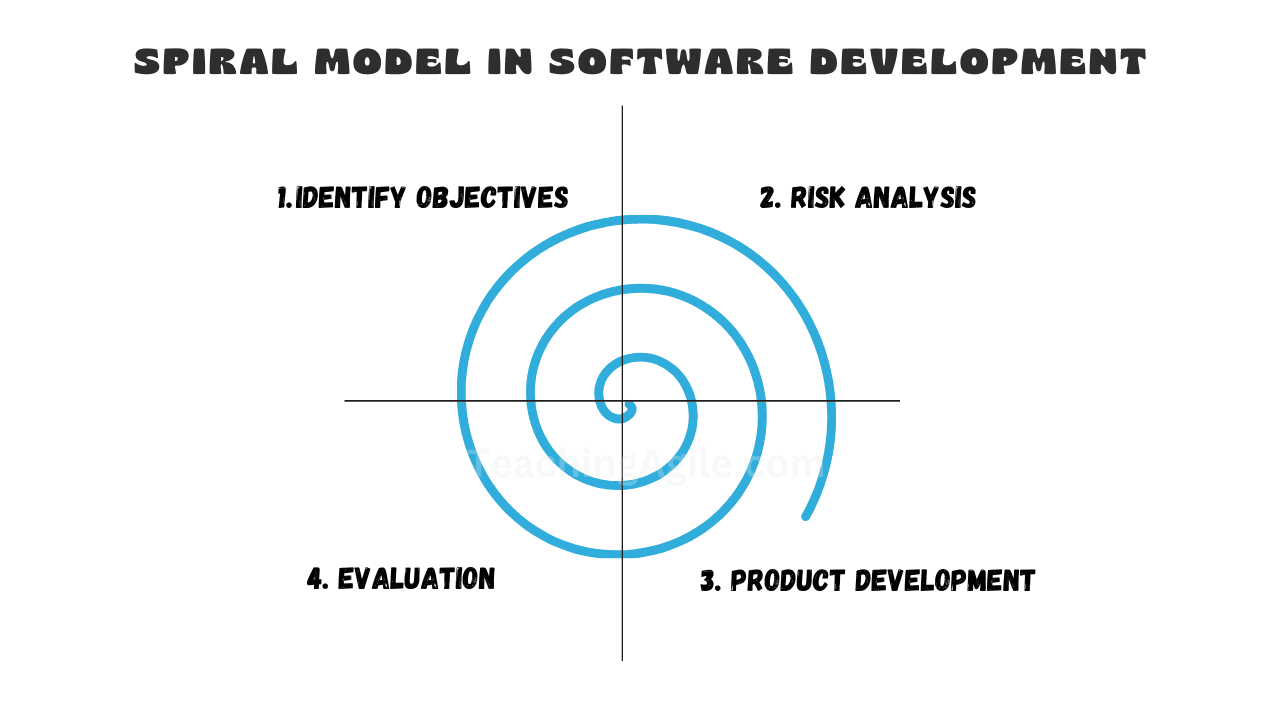
**Cooldown** – A time delay before a feature becomes usable again

**Persistent Data** – Data saved between game sessions

# Software Methodology

Software development has many different methods which are important to keep the project flowing consistently.

## Spiral Model



The Spiral Model was introduced by Dr. Barry Boehm in 1986 to reflect the iterative nature of software construction. The model is ideal for projects which evolve over time as it allows the refinement and adaption of ideas throughout the development process. (Talreja, 2024)

## Rapid Prototyping

Rapid prototyping allows developers to quickly check whether game features are possible and understand concepts and mechanics. Similarly, to The Spiral Model, it focusses on the iterative design process. The focus is on experimenting with different ideas, gathering feedback and refining the idea based on testing. The importance of this is receiving user feedback early in the development process and identifying potential problems quickly. (iXie, 2023)

## Chosen Methodology

Aspects of the Spiral Model and Rapid prototyping will be followed throughout the project. An important aspect of The Spiral Model is the planning phase with goals and requirements being defined. This is significant to make sure the project does not become over scoped and outlines potential risks with the project. Once the scope and risks have been considered the actual coding and integration and testing take place. (Talreja, 2024) This uses a mini-Waterfall approach which is a linear approach of designing, coding and testing. The waterfall approach by itself is limited due to the linear approach and lack of iteration which is why the Spiral Model has been chosen over the waterfall approach by itself. After the mini-waterfall approach an evaluation phase takes place to review the product against the initial goals with a focus on reflecting on customer feedback. This ensures the software meets what the target audience would enjoy.

Rapid Prototyping works hand in hand with the Spiral Model as the mini-Waterfall phase of the Spiral model is effectively rapid prototyping. A part of the system will be quickly tested to check the viability and try and identify mistakes early.

### Unity

Unity is the chosen engine for the mobile game product. Unity will make the project easier as a independent and scalable production as well as better fitting the requirements for the animation system and art style.

### Programming Languages

C# will be used with Unity to create unique behaviour scripts and databases for the project

### Asset Creation Tools

Paint.Net will be used to create the pixel art assets as png files as the program meets the requirements of being simple and fast to make pixel art. This will be imported into the Unity project.

### Behaviour-Driven Development

This focuses on what the software should do rather than how it does it and encourages communication with the user to make the software continues to meet the target audience requirements. (Wolfe, 2022) This works well with rapid prototyping as it focuses on running tests and refactoring code whilst ensuring the desired behaviours are maintained.

### Endurance Tests

Endurance testing has the purpose of evaluating the stability, reliability and performance of a system when subjected to sustained workload. The system is ran with a heavy load for extended periods of time to test the performance and stability. The tests are used to reveal issues such as memory leaks or performance bottlenecks. This is importance for the optimisation of the project which is significant with a mobile game where user retention will be lost if the software causes issues to the user’s mobile device.

# Coding Standards

## Naming Conventions

It is important to have coding standards to keep clean code which is more understandable for other people. The project should use upper camel case. No magic numbers should be used to make sure the numbers are modifiable to make iterations easier.

# Project Plan

## Scope

The project should release the minimum viable product to ensure the methodologies can be followed. For both the Spiral and rapid prototyping, working builds are required to evaluate. This also creatures a product for the main game to build off and allow for fast expansion.

## Milestones

1. Sprites added
2. Animation system added
3. Database of variables for the digital pet suitable for expansion
4. Behaviour tree which works with the data base and can change animations

### Milestone 1

1. Input System
2. UI to display database statistics

### Milestone 2

Basic functionality for increasing stats

1. Eating
2. Cleaning
3. Sleeping

### Milestone 3

1. Stats automatically depleting
2. Save system

### Milestone 4

1. Shop system
2. XP
3. Currency

### Milestone 5

Monetisation

### Milestone 6

Ready for future expansion

# Testing

Test driven testing will be used to produce working cleaner code as this method is proved to do. This happens in three steps, red, green, clean. Minimal code is written to understand how the system works and this may cause errors (e.g. red lines) then this is made to be able to pass and build and then finally it is cleaned up to fit coding standards. (Unadkat, 2021)

# Technical Design Documentation

This technical design document outlines the full technical architecture, component interaction and development decisions for ‘Virtual Baby Seal Pet’, a mobile-based Tamagotchi-style game. Built using Unity and C#, the design follows professional software principles, integrates design patterns and methodologies to ensure scalability, performance and player retention.

This TDD is built in alignment with the GDD particularly when referencing the game loop, stat systems and UI layout.

## Application breakdown

The code developed by the developer can be found in the “Scripts” folder.

## Input system

Registering inputs will take place from a Mono Behaviour script on objects which can be interacted with. This will check whether the user is touching the collider of said object then it will be check what gesture is taking place.

The system needs to be able detect swiping left and right on the seal pet to simulate a petting gesture. This should trigger a petting animation and produce goal and experience intermittently.

The system needs to be able to detect holding and dragging the food for the seal. This requires knowing when the player ends the interaction to cause the food to be dropped.

The system also requires the user to be able to throw a ball. This requires detecting a flicking motion and checking the amount of force added to make the ball move at the appropriate power and speed.

The system needs to check holding and dragging of the shower object. This should trigger a VFX system to simulate water droplets.

The system needs to check for a tap on the lamp object to toggle the light on and off.

The system will also need checks for taps on UI but this can be handled by the button systems in Unity automatically.

## Design patterns

Using design patterns is essential to keep clean code and also increase productivity. Furthermore, design patterns are important to keep the system expandable for future developments.

The DRY principle should be used to ensure the code is concise. The DRY principle standards for “Don’t Repeat Yourself” and the principle is that is there is a piece of code which is in multiple places this should be streamlined. (Muldrow, 2020) For example, the code could be put into a function so it can be called by two different functions or classes. Moreover, functions should perform one task only and fit onto one screen. This principle ensures clean code.

The flyweight pattern will be used to minimise memory by sharing as much data as with similar objects. Within this project this will be used by creating a single class to store all of the data for the seal pet and this will be shared with all the other objects. Therefore, every object can find out what state the seal pet is in. This seal data class can be used in future when there is multiple seal pets to design unique seals without creating a new script for each of the new pets.

The prototype design pattern will be used to avoid excessive subclasses and minimise the inheritance cost of creating new objects. Prefab objects will be used within Unity for any object which will need to be cloned so the objects can be easily instantiated. For example, there will be currency and experience which will need to be cloned so these will be prefabs.

The observer design pattern should be used in future expansion to add achievements to the game. This is because the design pattern holds a list of observers waiting for the message (e.g. that the achievement has been achieved) and sends the notifications.

The singleton pattern will be used for the Audio Manager to ensure only one global instance is accessed across scenes.

The State pattern will be used for the seal’s animation and behaviour states. This is useful when the seal switches between states such as eating, sleeping, hungry and idle. This will use a switch statement of the states to run functions of events which take place during each state.

The command pattern will be used for actions like feeding, cleaning or spinning the rewards wheel.

## Events

On start event will be used to initialise anything for the class that needs to happen when the app is opened.

The update event will only be used where necessary to avoid making too many processes happen each frame.

On State Exit will be required to change values and make various checks to make the states transition smoothly and cause relevant events to be triggered. For example increasing the relevant values such as hunger, energy and hygiene based on finishing the animations.

## Input Events

As the project is a mobile targeted application, traditional input detection will not be used. Instead it will be required to detect touch on the screen and in some cases certain motions. There will be a requirement for detecting touching many different objects, including the seal and items used to increase the needs of the seal. There should be detection of gestures such as swiping up, swiping across and dragging objects.

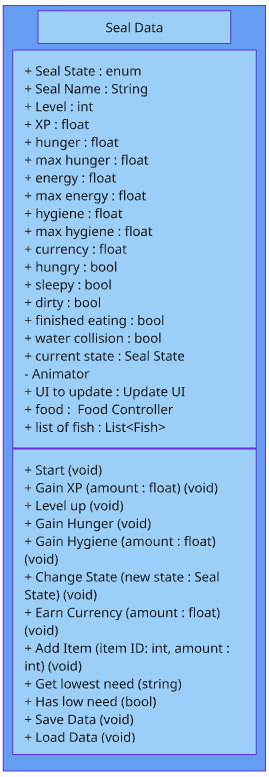
To create this system the type of “Touch” can be used to get where the user has touched. To detect gestures or to check for dragging, the end touch position will need to be found and the transformation between the two positions calculated. A switch statement can be used to check Touch Phases such as the beginning of the touch, end or detected movements. These phases can be used to trigger checking for the gestures at the correct time.

## Save System

The save system is required to save the data on the Seal Data class which consists of floats. Therefore, the save system can simply be made using the Unity feature of saving player preferences. However, to make the system more expandable encase in future there is more complicated data which is required to be saved, a JSON file saving system makes a better choice.

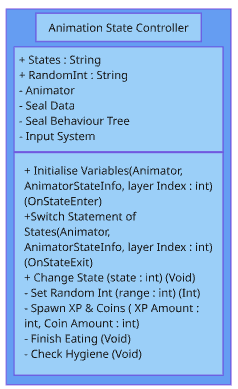
The system should be saved not every frame but when a piece of data is updated and when the application is closed. This is to avoid making saves every frame when the data is still the same. The data is loaded on the application start.

## UML Diagrams

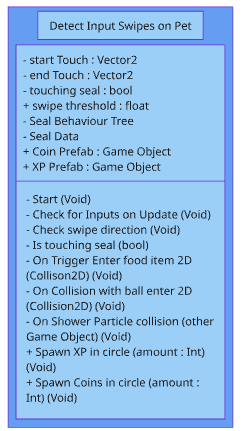
The Seal Data Class is the backbone of the system where all the data for the seal is stored, saved and loaded. This stores data for the seal’s needs such as hunger, energy and hygiene and has functions to increase or decrease these as required. The hungry, sleepy and dirty states have an additional bool as these states can happen simultaneously with any other state. For example, the seal can be eating but still be dirty. These bools impact the game in such if the dirty state is true the seal will always appear dirty using a particle system overlay. The get lowest need function is used when multiple of these states are true to check which one is the lowest and displays the animation for that state as a priority when the seal is idle.



The Seal Behaviour Tree Class is in charge of setting the state of the seal and reacting to what changes the state inflicts. There is also a function that checks the need stats of the seal to decide what state the seal should default to when an action is completed. For example, the seal finishes the eating action and animation, this class can check the stats of the seal, it finds the seal has a very low hunger need so the seal does not use the default idle animation but the hungry idle animation.



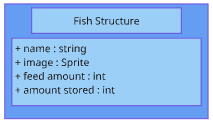
The Animation State Controller Class works with both the Seal Data Class and Seal Behaviour Tree to change to the correct animation that reflects the seal’s state or behaviour. At the end of a lot of animations an action needs to take place, therefore a state switch statement checks which animation just finished and fires off the appropriate reaction. The system can increase stats, for example the seal increases hunger at the end of the eating animation but also spawns some experience.



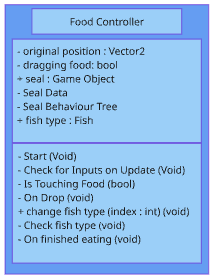
Along with the Seal behaviour tree, the Seal Data and the Animation Controller, this class is used to detect touches and gestures on the Seal object. This class can tell if the seal is being touched by the player and also check whether the player has swiped on the seal. Swiping left and right is used to trigger a petting animation on the seal.

As well as gestures, this class detects for collisions with other objects such as the food item: to trigger the eating behaviour, the ball: to trigger a playing animation and the water particles from the shower: to tell the game the seal is being cleaned.

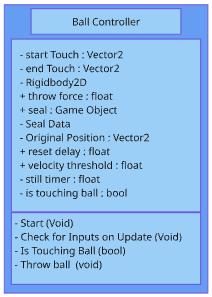
This class also spawns the experience and coins based off these inputs and collisions.



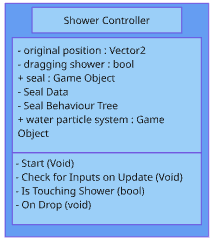
A fish structure will be used to store information about food items. This will be the image, name, how much it should increase the hunger bar and how much is being stored.



To increase the hunger stat of the seal there is a food item which can be dragged. It checks for a dragged motion and resets when dropped unless it collides with the seal’s mouth. It uses the information from the fish structure to work out how it should visually look so different food items can be swapped out. When the seal is finished eating the position should be reset. It also has a function to check which fish it currently is to allow for a check to be made to how much the hunger bar should increase by.



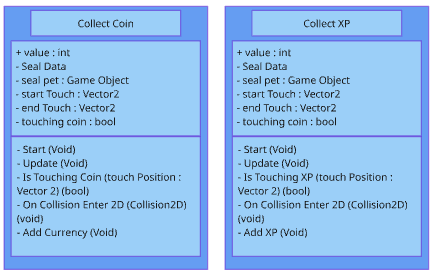
The ball checks for inputs and checks the force in which the player swipe the ball. This force is applied to send the ball across the screen. The ball will need a special material to make it bouncy and appropriate friction.



The shower checks whether it is being dragged around and on drop it resets position. It must also active a particle system whilst being dragged and deactivate it on drop.



The lamp must detect taps only and toggle the lamp on and off. This should visually make the screen dark and the sprite should change to and on or off lamp. Whilst the screen is dark the seal should go into the sleep state.

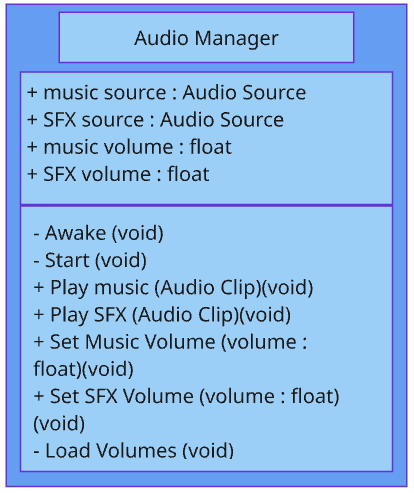


There should be coin and XP prefabs that can be spawned on the screen to award the player. When clicked on these should award the player currency or XP. They should naturally de-spawn if they fall out of the screen bounds.



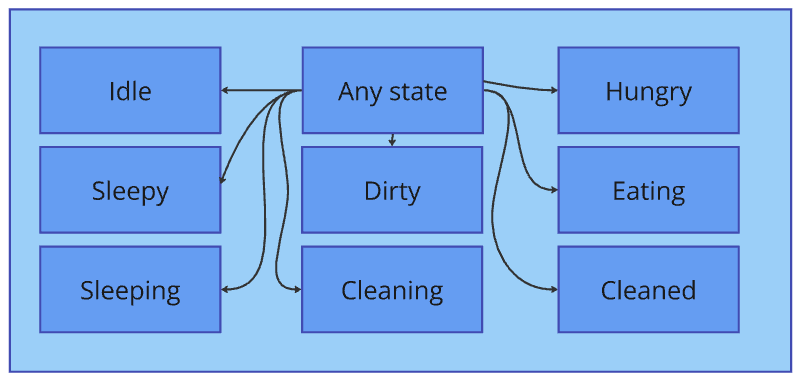
There should be a script to update the UI as stats are edited. This should include any text on the UI which is required to be updated. A function will be made that can toggle game objects on and off to allow the buttons to open and close the UI panels easily.

## Audio



An audio manager script will manage all the audio. It will be an instance, following the Singleton design pattern, which can be called from any script. It will allow the ability to play music and sound effects and change the volume. This will allow for a settings window to be created that can change the volume of the music and sound effects. It should be able to save and load these settings using player preferences.

## Animation System



The animation system will use the animation state machines built into Unity. Each animation will have an entry condition of a particular number and an exit condition that the entry condition is no longer true. Each animation needs to be accessible for any state as the animations can play in any order depending on how the player decides to interact with the seal. The entry and exit conditions also make the animations smoothly blend as the animations will not have to have finished displaying before moving to the next state.

The Seal Behaviour Tree Class has a switch statement which when a state is entered the variable for the animation system is changed. This in turn smoothly changes the animations with the seal states.

# References

iXie (2023). *The Growing Significance of Rapid Game Prototyping in Game Development*. [online] iXie Gaming. Available at: <https://www.ixiegaming.com/blog/significance-of-rapid-game-prototyping-development/>.

Muldrow, L. (2020). *What is DRY Development? | DigitalOcean*. [online] www.digitalocean.com. Available at: <https://www.digitalocean.com/community/tutorials/what-is-dry-development>.

Talreja, A. (2024). *SDLC: Exploring the Spiral Model and Its Benefits – Nextra*. [online] teachingagile.com. Available at: <https://teachingagile.com/sdlc/models/spiral>

Tricentis. (n.d.). *Endurance testing in software: What it is and examples*. [online] Available at: <https://www.tricentis.com/learn/endurance-testing-what-it-is-and-examples>.

Unadkat, J. (2021). *Test Driven Development (TDD) : Approach & Benefits*. [online] BrowserStack. Available at: https://www.browserstack.com/guide/what-is-test-driven-development.

Wolfe, M. (2022). *Examining Behavior-Driven Development - CodeX - Medium*. [online] Medium. Available at: https://medium.com/codex/examining-behavior-driven-development-5db006edfee5 [Accessed 29 Mar. 2025].

‌

‌