Mobile Application Development Practical Assessment

System Design:

Game v1:

Game architecture and main components:

The game scene has a spawner that at a random range will spawn a new flying object.
The spawner will randomly choose how far the object will move in the scene (x and y).
And randomly choose the type of object, being a rock or one of three fish types.

A player touch detection is also present. It detects whether the player is touching the screen, and shows a fishing net that the player's finger is "holding". It also determines whether the player has been swiping upwards in the last few frames.

The fishing net has a collider attached that triggers each time a flying object interacts with it. It checks if the player has been swiping upwards. If they have the object is checked if it is a rock or a fish, and removes a health point or adds a score point respectively.

When health equals zero, the HealthManager script will spawn and overlay the GameOver scene on the play area. It will change the playState in GameManager which the player touch detector will listen to and destroy itself so the player can no longer catch fish. UI buttons do not need the player touch detector.

How data is managed:

Fish types enum, game state and the game's run time (which runs fish spawn delays) are stored in the GameManager script, which is loaded globally (Singleton). This allows them to be access by all scripts easily for assigning and checking.

The GameStart script gets the screen size once per run and calculates different paddings for UI components to space themselves accordingly. This script is attached to the root of the game scene to be easily accessible through this path $"/root/Game".

Most other data is managed through the Spawner script. Each sprite and the flying object prefab (called fish at this time) is preloaded, so they are loaded once per game launch. Spawner calculates the min and max distances (x and y) that objects can move based on screen size. The spawner also manages the bomb (rock) spawn chance. It finally holds the last spawn time and the delay until the next object can be spawned.

Health and Score are held by their own Manager scripts. Both having an update value method that allows easy changes to be made to both the value being stored in the script, and the visual display for players.

Key design choices regarding scalability, maintainability and performance:

The flying object is a prefab, which means it can be added into a scene and will work immediately. I change the object’s type, sprite and path points to fit when spawning.

The flying objects will despawn immediately on completing their assigned path. This means the performance of the flying objects should always be the same regardless of how long the game has run.

The game is maintainable by ensuring scripts fit a specific scope, and are as short as possible. Assets, scripts, scenes etc are saved in specific folders in the file system to be easily found and have descriptive but short names.

Game v2:

Game architecture and main components:

The spawner, player touch, fishing net and health manager all work the same as v1.

The SaveData script runs the Save function for each script added, and saves the dictionary returned as a JSON file. This script can also load and delete the data that has been saved.

The main menu has access to a tutorial. Which has some simple animations and words explaining how players can catch fish and rocks, and which is good or bad.

The main menu and game over scenes allow the player access to their collection. This is a simple grid of each object that can fly across the screen. Although if you've never caught it before the sprite is a silhouette and the name is obscured. Once players have caught at least one, the sprite and name for that object type will show. Also showing a counter for the number of times that each object has be caught.

New collectables after each run are shown on the game over scene, so players know to visit their collection to see the new additions.

Players can see their highscore on the main menu and game over scenes. An animated banner will show when players get a new record.

A timer showing how long the current run has last shows in the game scene.

How data is managed:

GameManager still holds the object types, run time and game state. But now it holds the flying object stats resources (including sprites which have been removed from spawner), collectables counters, and highscore. This is to keep all these assets available to all scripts as easily as possible.

The SaveData script doesn't hold any data, but allows the device the game is played on to hold collectable counter and highscore is storage. This allows these values to persist between game launches.

Flying object stats are now in resources which allow for easy data accessing and changing in editor if needed.

Key design choices regarding scalability, maintainability and performance:

Using resources allows for easy scalability. All being loaded into the game in a single place, GameManager, also allows for easy implementation.

The game is kept maintainable by keeping scripts short and to following SOLID principles.

Performance was though of when making the water splash effects. I reduced the number of particles to three and gave them a short lifespan so there are reduced number of assets loaded at any one time.

UI Design:

Game v1:

Responsive design for different screen sizes and orientations:

The game calculates the screen size each time the game scene is run, and this allows the UI elements to pad correctly to each other. It also adjusts the max pixels that the flying objects can move, so they can cover most of the screen. Sadly, it doesn't change the scale of the UI elements.

The background image is set to cover the screen, so regardless of the size, it will always cover the whole thing.

The UI in the game over scene is centered both vertically and horizontally. This allows it to fit to any screen that is larger than 600x600px

UI/UX decisions, including navigation, interactions and usability:

The game over scene has a button to play the game again. This button has a pressed effect that makes it slightly smaller and darker.

All text in the game has a background that allows it to be easily read no matter what is behind the fish, be it the background image or a flying object.

Wireframes or mockups for key screens showing layout and navigation:

(see the wireframes in v2 below)

Game v2:

Responsive design for different screen sizes and orientations:

Main menu's UI is anchored to the center and corner, allowing a min screen size of 950x600.

Tutorial and Collection UI elements are anchored to the middle of the screen.

Game scene calculates the screen size to place UI effectively, and adjust the fish jumping zone to fit. The UI elements are anchored to the corners. Minimum screen size of 850x600px

Game over's UI is anchored to the center and corner, minimum screen size of 880x680px

UI/UX decisions, including navigation, interactions and usability:

All UI buttons only activate if the player presses down on them, rather than lifting up. This stops accidental returning to main menu on the game scene with players swiping upwards to catch fish.

All UI buttons have a pressed effect, making them smaller and darker.

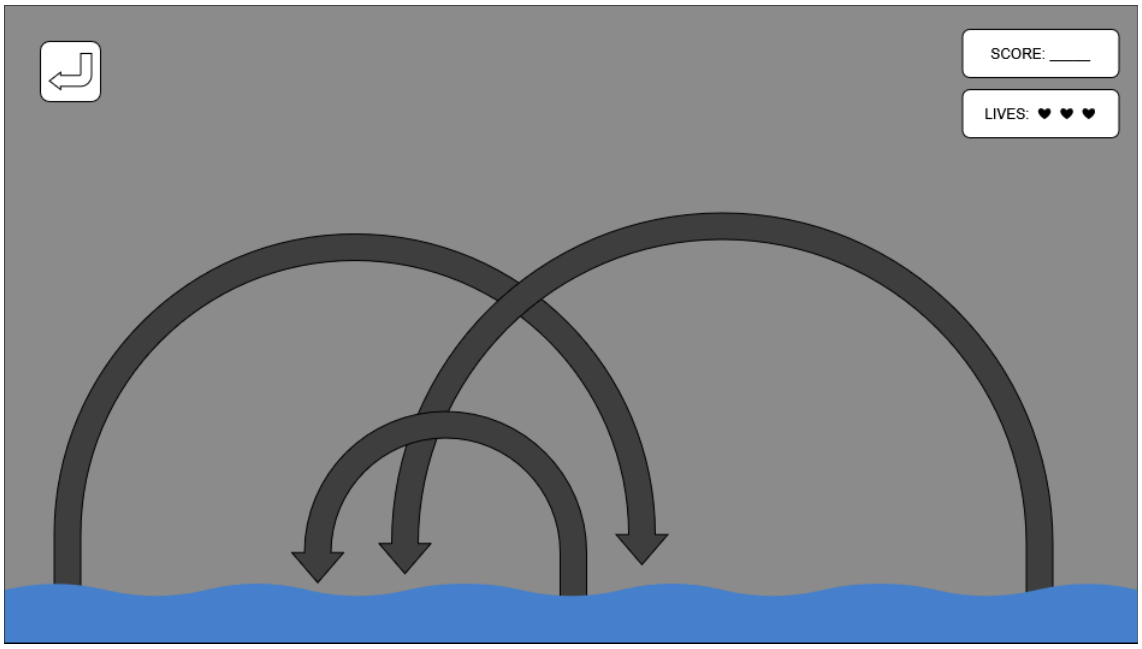
The delete save button has a confirmation window, with "yes" being the furthest away from where player's finger just pressed. It is also off to the side away from the main buttons to press, in a potentially awkward position if the player is using their thumb. These all to stop accidental deletions.

Animations in the tutorial scene loop automatically, so players don't have to keep pressing a button to see the animation. They also have a small pause before repeating to not overstimulate players

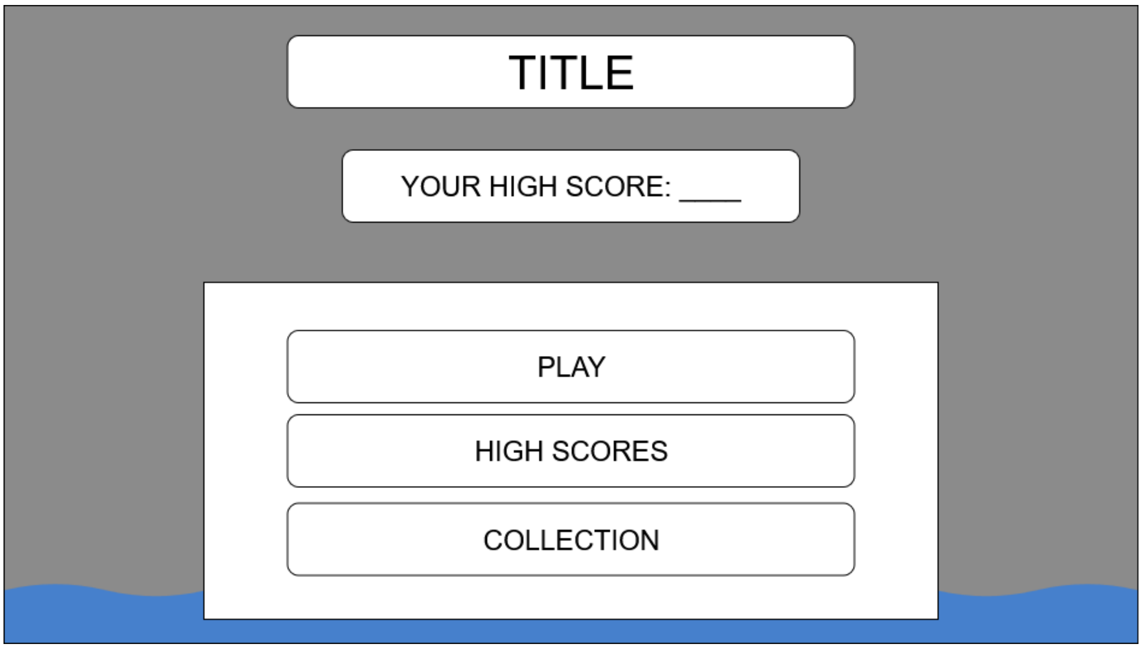
I kept convention with the placement and icon on the return to main menu button that is in several places in the game. They are the same as each other, and the same as other apps that allow you to return.

Text in the collection has an outline to allow for easy reading of potentially small text, that may have different background colours, depending on the object's sprite.

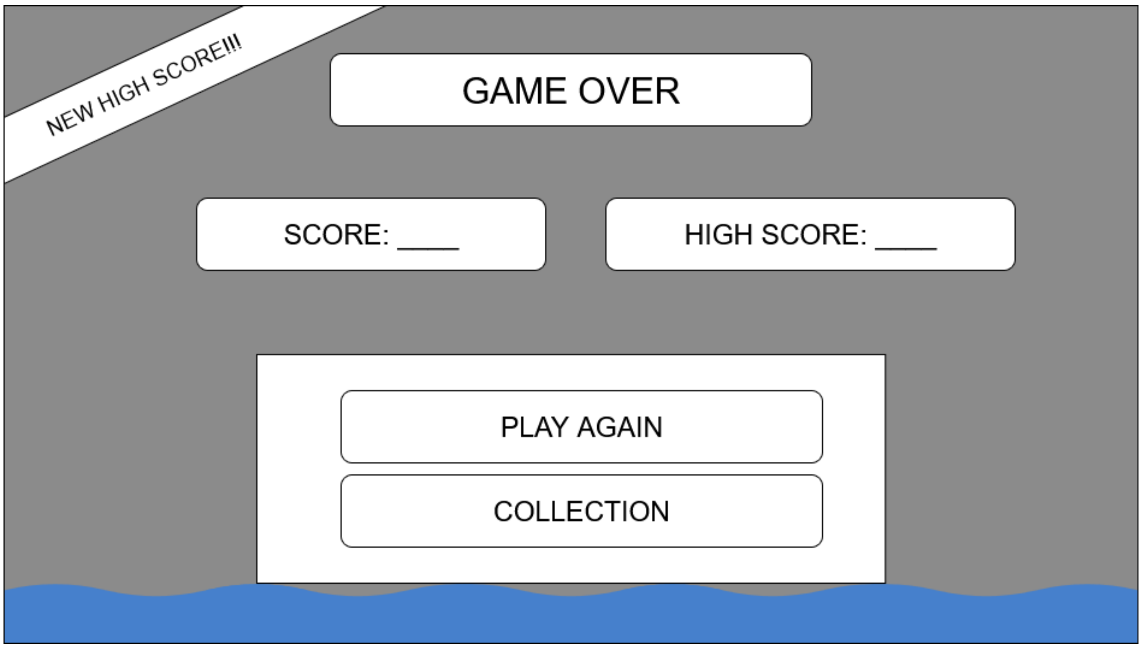
Wireframes or mockups for key screens showing layout and navigation:



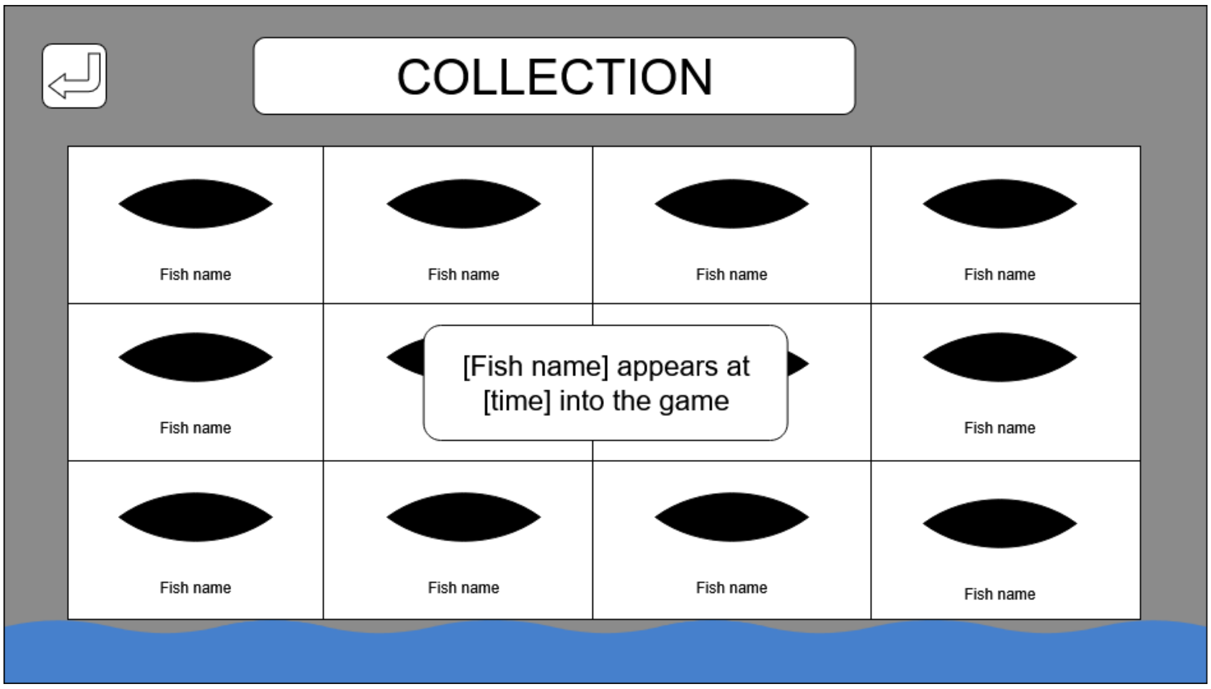
Game Scene
The main game section has a return button that allows the player to go to the main menu. Score will be a number. Lives will show as 3 heart icons that each icon is made invisible as health as decreased. The blue waves indicate a waterline that shows where the fish are coming from. The large arching arrows indicate the area where the fish and non-fish items will be. They will leave the waterline and remain in the air for a while, then return to the waterline.



Main Menu
The main menu will show the name of the game and have 3 buttons for players to tap.
The background will be the same as the game scene, including the waterline



Game Over
Game over screen will show the last runs score, and the previous high score. A banner at the top left will show if the player beat their previous high score. Buttons to play again and view the fish collection are able to be tapped.



Fish Collection
Each section of the grid will have the fish's name and sprite. Players can tap on any fish for a pop-up to show when the fish appears. When the fish is undiscovered the sprite will be only a silhouette, after collected, the fish will be the normal sprite. Return button goes to main menu.