Reef Defender #1

Client:The Giving Child

Authors: Niall Miner, Edgar Gutierrez, Brian Yoon, Henry Dau, Charles Widdicombe

June 16, 2017

COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

Table of Contents

1 Introduction

1.1 Description of Client

The Giving Child (TGC) is a nonprofit organization focused on empowering kids to make a difference in fun ways. In practice, this means creating fun game apps that teach children about crises around the world while simultaneously raising money for nonprofits from game purchases and donations.

In the last two years, the percentage of kids using mobile apps jumped from 38% to 72%" (Common Sense Media). Game apps are an excellent way to appeal to kids and to help get them involved. The goal of TGC is to create game apps that are both fun and informative to allow young children to become aware of some of the world's biggest issues. Game apps allow an organisation to potentially reach a very large number of people while also being an effective platform for raising funds. According to Newzoo, a market-research firm, "mobile games will hit $30.3 billion in 2015". The mobile app market has an impressive ability to influence the world positively, and will only grow in the future. Game apps are a parent's new babysitter, a kid's new toy, and the world's chance for change. TGC is helping to raise awareness of important events in schools and mobile media, through events, fundraising, and competitions. They are developing fun and engaging games that are designed and built by gamers and students with passion for games. From the purchase, 90% of profits go toward a nonprofit related to the game's cause and the other 10% of profits go toward the building of TGC's next game. The child begins playing an educational game that showcases the chosen cause in an upbeat and informative way.

TGC's current game app will be focused on teaching kids about the effects of pollution on the ocean and how they can help be a force for good in cleaning them up. We at The Giving Child view pollution in our oceans as a serious crisis because we've seen first-hand how sea life is being negatively impacted. We hope that we will be able to make a positive impact on our world's oceans with this project.

1.2 Vision

The goal of the Ocean Game group was to create a mobile app game for toddlers and kids that helps them learn about the importance of keeping our oceans clean. Since this application was split up into two teams, our team was responsible for creating the main framework for the game, which includes the menu system and the basic level layout. The game was intended to be centered around cleaning trash from the ocean. We were asked to create a game that would be simple and enjoyable for children to play. The game needed to be mechanically intuitive and appealing to children in appearance.The menu system had to be simple and easy to navigate, while looking clean and professional.

2 Requirements

2.1 Functional Requirements

- The game will start with two difficulty levels--one for toddlers (under 4 years) and one for 4yrs and up.
    - Toddler level should be harder to exit the game (client initially wanted the home button to be altered, but this is not a viable option--agreed to pursue different options)
- Main menu that allows for difficulty selection, game options, and a donate page
    - Donate page can be reached from the menu and will describe benefits of supporting the specified charity and give a link to their donate page
- Two defined levels
    - Level 1: tap floating trash to collect it and clear from the ocean
    - Toddler Level: an easier endless version of level 1 where toddlers can simply have fun without consequence

2.2 Non-Functional Requirements

- Game options will include sound options and maybe more to come
- Android and/or iOS game
- Built using Unity
    - Unity was decided on 5/16 as the platform of use because of its ability to port to Android and iOS and because of its functionality as a game development tool
- Release to App Store and/or Google Play
- Original artwork or public assets
- Sound effects and music to create a more stimulating game

3 Architecture

Our game was set up so that from the menu screen, one has the choice to navigate to the various difficulties of the game, the "reef" or aquarium space, and the about page. The game consists of a single level in which the player collects trash by tapping it. The difficulty levels consist of an increasing number of the types of game objects that would appear, as well as an increase at the rate at which all objects spawn. The reef consists of a number of scenes in which the player can view the special fish that can be collected occasionally from the main game. These fish all have different behaviors and can be named. The about page contains the game's credits and attributions,as well as a donate button which navigates to the client's donate page. The menu also includes a button that goes to a page explaining how to play the game.

The game was split up between two groups, with one group handling the menu and basic game functionality, and the other group developing the aquarium mode and most of the higher difficulties' more complex features. This did however lead to something of a handoff situation between the two teams.

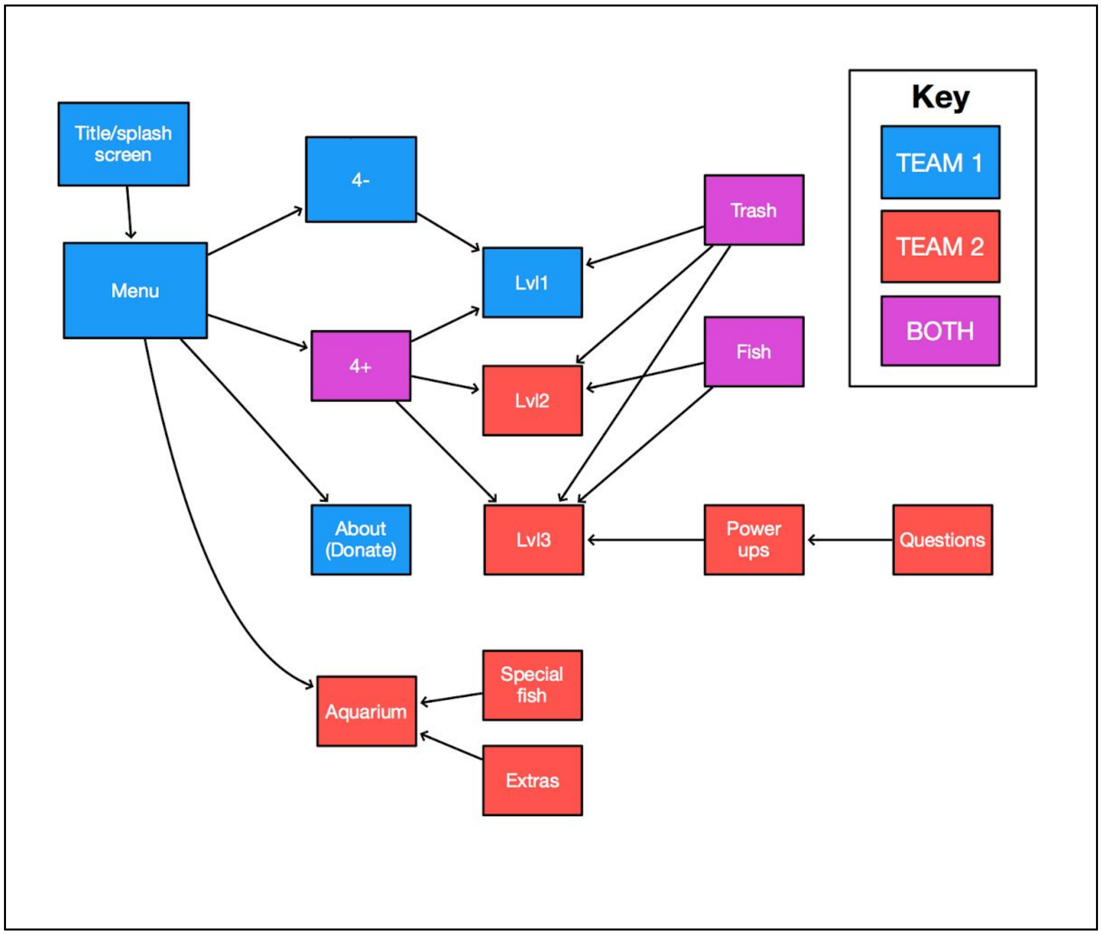

Figure 1: Table of Responsibilities
This figure shows the breakdown of work between the two teams. Since there are two teams working on the same project, the teams had to decide who would be doing what. Above shows the results of that breakdown. In broad strokes, Team 1 is responsible for most of the front end and the basic game. Team 2 is responsible for adding difficulty to the game and the reef (aquarium) to store the special fish.

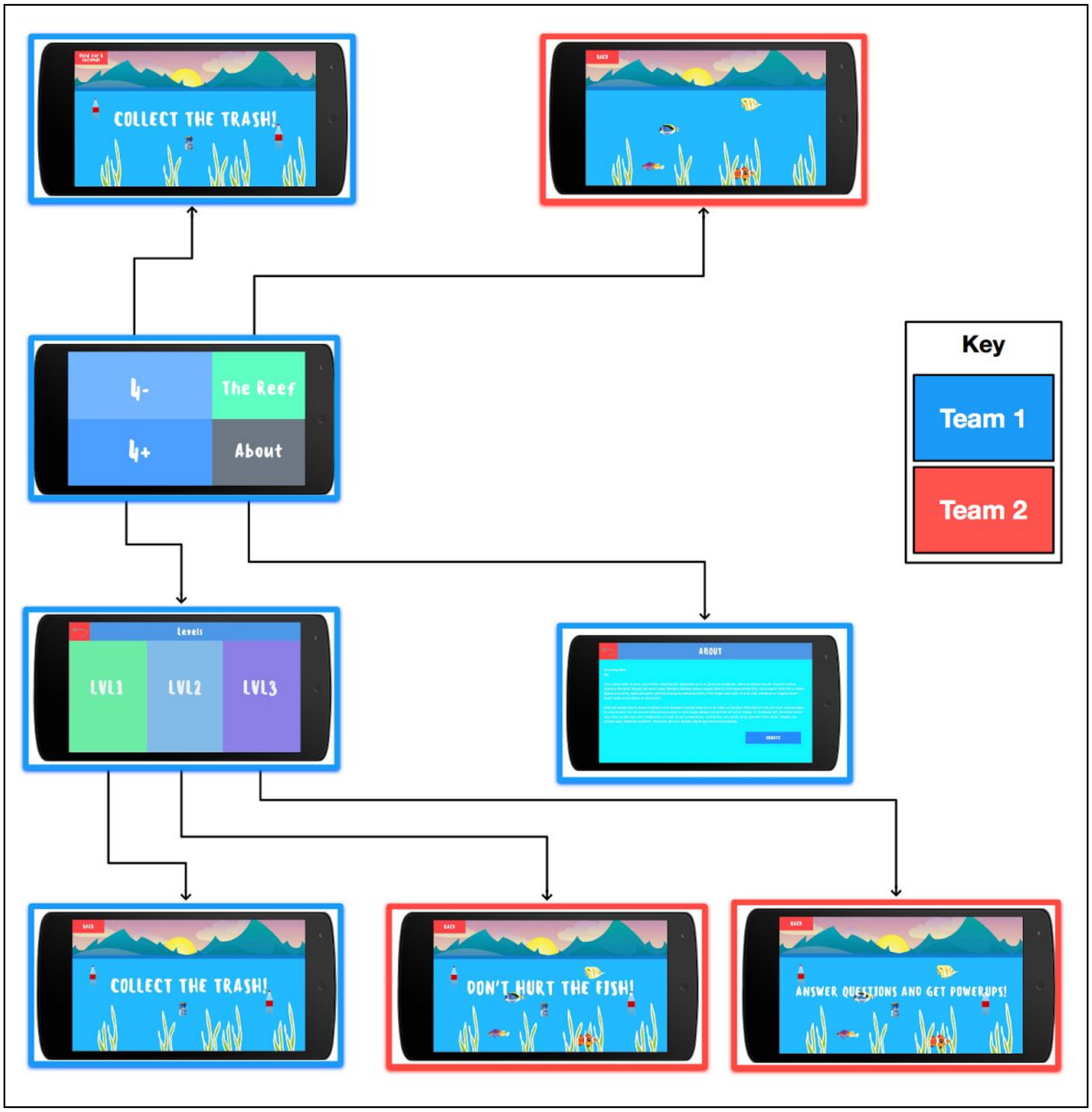

Figure 2: Storyboard

This figure shows a more graphical breakdown of the workload. It shows much the same information as Figure 1, but in a pictorial way. Notice Figure 2 still represents the work breakdown of the final design. A special note is that this is an introductory storyboard and does not represent final art decisions.

4 Technical Design

4.1 Use of the C# Programming Language

Building the game and implementing it correctly required more than creating game objects and adding components using the Unity user interface itself; it also required the use of C# to dictate game object behavior using Unity library extensions. Within Unity, game objects get C# scripts "attached" to them as components, and any functions used is those scripts reference the object using it. For example, within a script, the Start() function determines how an object gets initialized, and the Update() function is used to know where the object should be in the next frame of the game. Both functions are defined by the user. Additionally, the same script can be attached to different objects that need to accomplish the same task. In our case, this is how the special fish's swimming behavior was implemented. To demonstrate how this behavior works, Figure 3 below shows a snippet of our Swim.cs code.

```
1   // Update is called once per frame
2   void Update () {
3
4       if (!GlobalVariables.isPaused) {
5
6               Vector3 pos = transform.localPosition;
7               if ((speed > 0 && pos.x > X_BOUND) || (speed < 0 && pos.x < -X_BOUND))
{
8                       if (keepSwimming) {
9                               speed *= -1;
10                              transform.Rotate (Vector3.up, 180);
11                              pos.y = (float)(.5 - Random.value) * Y_BOUND;
12                      } else {
13                              Destroy (gameObject);
14                      }
15              }
16
17              transform.localPosition = new Vector3 (
18                      pos.x + (float)speed / 10,
19                      pos.y,
20                      pos.z
21              );
22      }
23  }
```

Figure 3: Swimming Behavior

As shown in the figure, Update() is called once per frame, and it is called for each object currently on-screen. The function first checks if the game is currently paused (line 4), since fish should only swim when the game is active. The variable pos, (line 6) is used to first copy the fish's current position, and is then modified to hold the fish's position in the next frame. This is accomplished by making sure that the fish is within the screen bounds (line 7). If it is, then it just keeps swimming, moving a certain distance dependent on its current speed (lines 17 - 20). If it is on or beyond the bounds, however, two things can happen: If it is a fish that should keep swimming (line 8), its velocity is reversed (line 9), its image is flipped (line 10), and its y position on the screen is randomized (line 11). Then, its next frame is determined as described earlier. Otherwise, if the fish should not keep swimming (line 12), it is removed from the game (line 13).

The functionality of the scripts with their corresponding game objects is similar to the relationship between abstract classes and concrete classes in the C languages; a C# script in Unity describes functions that all objects that use it must adhere to. This is also similar to the use of interfaces or abstract classes in Java. Using scripts and not relying entirely on a user interface allows programmers to give precise instructions to their objects, and it is this kind of flexibility that allowed us to make the game with the specifications given to us. Coding in C# also provided a sense of familiarity, in contrast with Unity's working environment, which was new for our group.

4.2 Adding Animations

The animations used within the game were implemented using a stop-motion approach. Once official images were decided upon, we used a program called the GNU Image Manipulation Program (GIMP) to tweak the source images and create the frames necessary to give the impression that the fish and other features were moving naturally in the ocean. Specifically, we used GIMP's perspective tool to slightly bend the images during every frame. Most animations used five unique frames to create animations that were nine frames long and would continue to loop. Figure 4 gives an example of how this was executed in Unity.

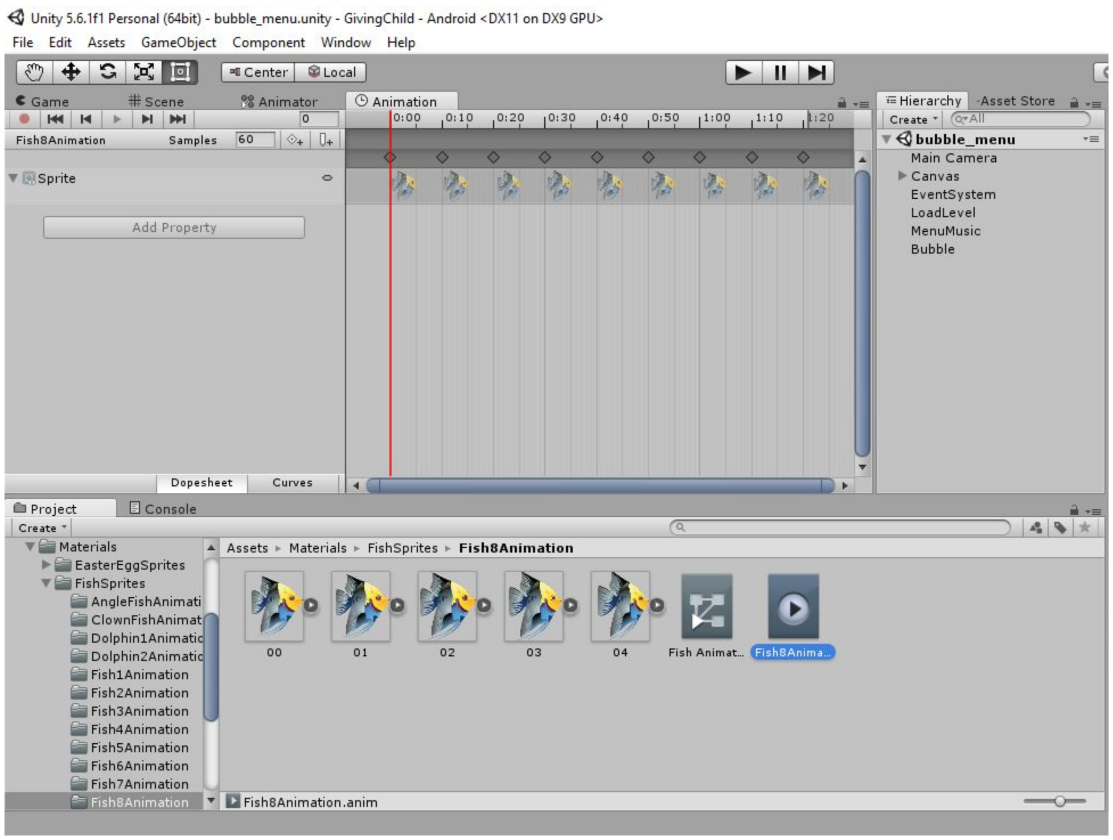

Figure 4: Fish Animation Example

The example in the figure is for one of the fish in our game. The five basic frames used in the animation are shown on the bottom while the frames used in order are shown in the animation window. Frames 00 and 01 correspond to the fish moving its tail away from the viewer, frames 03 and 04 correspond to the fish moving its tail toward the viewer, and frame 02 is the unchanged original image, representing a neutral position. The order in which the frames are used in the animation are as follows: 02, 03, 04, 03, 02, 01, 00, 01, 02. Using this order creates the effect of the fish moving its tail-end back and forth as it swims through the ocean. A similar approach was used for the other fish, with a few exceptions. The kelp animation used a different kind of perspective tweaking of its original image, but the basic approach was the same.

While other, more efficient methods exist to produce 2D animation in Unity, using our frame-by-frame method proved to be the way that made the most sense with our limited knowledge of the platform. The process was relatively easy to explain to teammates, and it also allowed us to have a better sense of control over the animation properties, such frame usage and playback speed.

5 Decisions

Our group's technical design decisions were made primarily on the use of software due to the nature of our project. The client requested the development of a mobile game for young children with an aquatic theme. The basic features of the game include simple tapping and swiping, and it has no necessity for far more complex features. Hence, when looking for an appropriate integrated development environment (IDE) for building this game, we were exploring options that would not be as taxing on hardware while at the same time providing flexibility with game mechanics. Additionally, we needed an IDE that was able to not only port the game to both iOS and Android devices, but also allow us to create and distribute the game with little to no licensing restrictions.

We initially agreed upon GameMaker Studio for our development environment. It provided an interface that allowed an efficient workspace for 2D gameplay, which was ideal for our game ideas. However, this IDE had both portability and licensing limitations, which would hinder development in the later stages of the project. Thus, we decided to search for other options and ultimately decided to pursue the project using Unity.

While Unity provides more advanced features that were ultimately unnecessary for our relatively simple aquatic-themed game and is overall more complex than was needed for our project, it allows easy portability for both iOS and Android devices. We used C# to code the scripts for our game objects, and while our group has not worked with this language before, we have had extensive experience programming in various C based languages. Of the languages that have supported Unity libraries, C# is the only C based language which led us choose to use it for our project. Furthermore, we ended up extensively using Unity's C# libraries in our scripts, meaning that we were mostly learning how to use Unity's libraries in our time working on the game as opposed to the language itself.. While it took some time to learn how to properly build in Unity, the results seem to indicate that we made the right choice. We also felt that becoming familiar with the Unity development would be a useful skill to have for the future.

Since the game idea was simple to start with, we've had many other ideas that we thought would fit nicely into the game's structure. Since we ended up using Unity to develop the game in, we are able to easily implement many of the new ideas we've had to make the game more robust and feel legitimate according to our client's needs and values. We've also had some cutbacks to our application due to internal functionality of the devices we are porting to. Since we are limited to Android and iOS phones and tablets we are not able to modify what the home buttons do, so changes to how our app was closed had to be applied.

We chose to include a game mode in which the player can view the fish hey have collected by playing the other game modes. This mode was not included in the requirements provided by the client, but we felt that providing an incentive for the player to play the game outside of points would help improve the longevity and replayability of the game.

A smaller technical design decision that was made was to use more collision boxes (similar to Riemann Sums) so that collisions with things such as the rock are more accurate. This is not that crucial of a decision but it still affects how the game works visually. Before this change, we had dust particles appearing where the original box was, but this was not accurate to where the rock was.

Another decision was made was to limit the game to only be in landscape mode. This provides more coherency between devices and allows a fuller use of the game area rather than in landscape.

6 Results

Overall, we feel that we met the requirements given to us by the client, as well as completing the game to our own satisfaction. The playable levels meet all requirements given by the client and the menu system navigates to appropriate areas of the game. Testing of the app has been done throughout the duration of field session (approx. two months) on both Android and iOS devices. Both devices have resulted in successful results with no issues in functionality. More testing at this point of time is necessary for large iOS devices (i.e. iPads) as we foresee many children playing the game on tablet sized devices. Figures 5 and 6 that follow show screen-shots of our in-game menu and one of our levels.

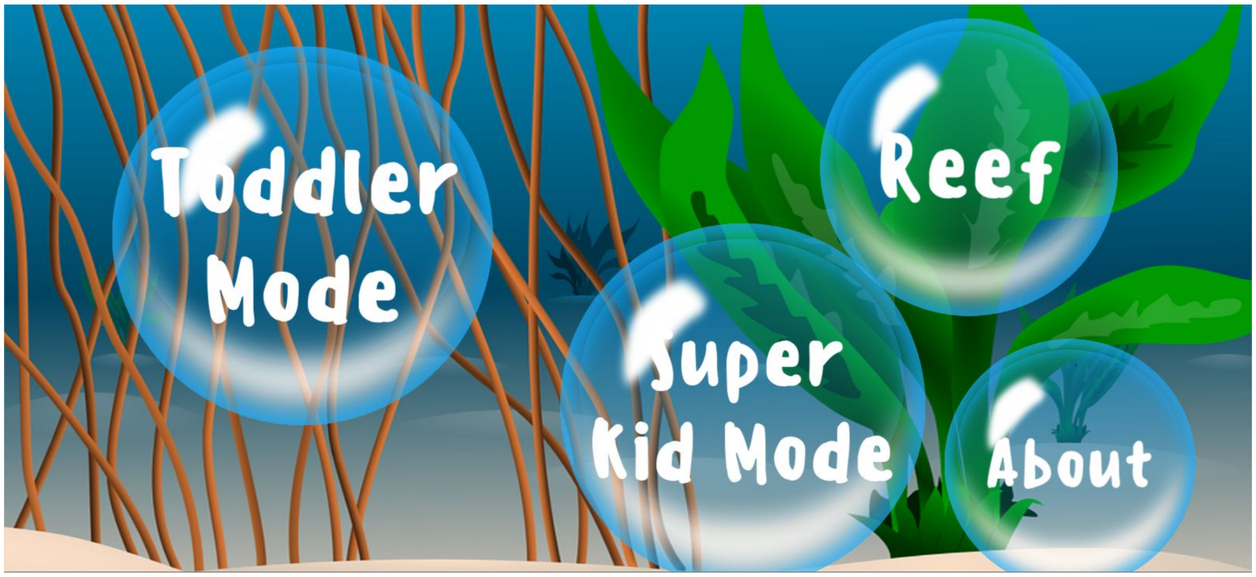

Figure 5: Main Menu
The main game menu, fitting the aquatic theme with the use of bubbles.

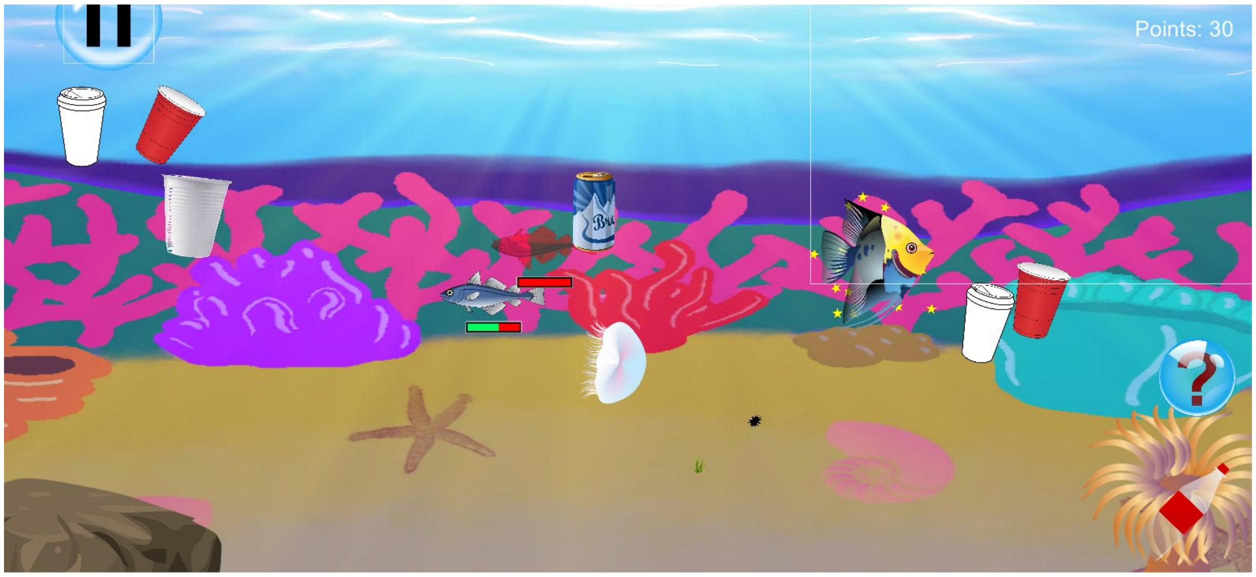

Figure 6: Game Screen Capture
A captured image showcasing game features such as trash collection, collectible "special fish," and question bubbles.

6.1 Features Not Implemented Due to Time Constraints & Future Work

There are a few items that can be addressed in future work but are not necessary for the functionality of the app. For example, there is a time lag when loading a level of about 2-3 seconds. A loading screen or animation may be added to increase the fluidity of the UI..

We had hoped to include a fish facts page for each of the fish found in the reef, but due to time constraints this was never implemented. Similarly, we had entertained the possibility of making certain aquarium fish only be able to be collected in certain game modes or at certain point thresholds in order to encourage replayability , but this was eventually abandoned in favor of a simple probability based spawn table.

6.2 Testing

Once certain milestones were passed during development, the game was compiled and ported to appropriate devices (Android and iOS) to test game variables, such as hitboxes, element placements, overall visibility, and playability. During each playtesting period, we collected feedback from our own experience with the current build as well as feedback from a few select other playtesters. Using this feedback, we tweaked or reworked features to improve the game's overall experience. We started off with just Android testing because it was easier to port over, but after searching through Unity settings we were able to port the app over to iOS for even more device testing. Since almost all of us have different devices we had ample testing opportunities.

6.3 Lessons Learned

Over the course of this field session, our group has primarily grasped an intermediate understanding of the Unity working environment. Additionally, we have gained experience in general game design and mechanics with regards to scripts in C#. This project provided insight into how a development team tackles a solution to a client's request, as well as the speed bumps that may arise on the way. Finding ways to efficiently accomplish the required tasks was the greatest challenge and provided us with great opportunities to overcome obstacles.

We also learned a great deal about handling a project with multiple groups and a large number of individuals. We learned that it was important to assign roles and responsibilities to our groups as soon as possible and with as little openness to interpretation as possible. We ended up spending most of the first week addressing this issue. Once we had done that it meant that the different portions of our game could be developed independently of each other and merged at a later time.

Appendices

A.1 Attributions

The music for the game was provided by Kevin Macleod and Incompetech.com.
Sound effects provided by Freesound.org.
Images provided by PixaBay.com and clipartall.com.
Font provided by Ezequiel Ergo at behance.net/eergohiki.
All assets used under a Creative Commons Attribution License.

A.2 Figures