Water Hero

The Giving Child

Harrison Oest, Kevin Barnard, Tiger Kheng,
Joseph Thurston, Ryan West, and Abhaya Shrestha

Table of Contents

Introduction

What is The Giving Child?

The Giving Child is a non-profit organization started by Asha and Evie Barnes that aims to empower children through mobile applications and games. The games that they create are meant to educate children about issues around the world (unsafe drinking water, food deserts, polluted oceans, etc) and show them that they can make a difference. The Giving Child has been a part of the Computer Science field session for the past several years, giving students the opportunity to help out with their cause and to gain experience in mobile application development.

Project Overview

Many communities around the United States have water systems that are getting older and more dangerous. The danger comes from the presence of lead and other elements unfit for human consumption. Young children are at the biggest risk, suffering the most from unsafe drinking water.

This problem is unlike other humanitarian problems in the fact that there is a well-researched solution. Replacing aging pipes is the best primary option, and supplying filters or bottled water are good temporary alternatives. The Giving Child has proposed we make an iOS game to help educate children about contaminated drinking water. The target audience for this application is privileged children who do not understand that some people in the United States cannot walk up to the faucet in their kitchen and have clean, safe water to drink. Therefore, our overall goal for this project was to develop a game in iOS that is playable and educational for little kids ages four to eight. Our goal with Water Hero is also to instill sympathetic and empowered feelings for those suffering from this problem and raise awareness of the need for maintenance on these damaged and outdated water systems. In addition to our overall goal, The Giving Child hoped to use the money made from the app to support the cause.

Requirements

From our client's perspective, the game is a 2D Side Scroller with water droplets moving from the right side of the screen to the left. There are blue water droplets representing clean water, and red droplets representing contaminated water. The user taps on the red water droplets, causing them to disappear. This signifies that the contaminated water is being cleaned. When the red water drop is tapped or missed, it causes a counter to increment or decrement by one respectively. The level is over when the required amount of drops for that level has been collected. Additional functional and nonfunctional requirements from our clients are listed below.

Functional Requirements

- Has a home page when the game is started.
- The game needs to have background music.
- On the home page, there is a dial to decide the speed of the game.
    - The dial goes from 1 - 30 with 30 being the most difficult. The client wanted the dial to go from 1 - 30 because of the *Thirty...Speed* video on YouTube of a child she found adorable.
    - Additionally, the speed functions by making the drops spawn faster and move faster.
- The menu button pauses the game.
- The user cannot exit the game unless the user holds the on-screen menu button for at least 10 seconds to prevent the player from accidentally exiting the game. This functionality is only applicable for lowest speed for toddlers. The reason for this is because our clients do not want the kids to accidentally press the pause button every time. That way, the parents do not have to press play again and again.
- A counter pops up and counts down from 10 to 1 when holding the menu button.
- There are 3 levels in the game.
    - In level 1, the user needs to tap on the water drops to destroy it and have the counter go up. A total of 150 water drops are required to win this level (150,000 people affected by the crisis divided by 1000).
    - In level 2, the user needs a total of 250 water drops to win this level (250,000 divided by 1000).
    - In level 3, the user needs 19,000 water drops to win this level (signifying 19 million divided by 1000 water drops).

Non-Functional Requirements

- The application is made for iOS. Therefore, the Swift Programming Language is used
- The game is playable and easy to understand for a 3-year-old.
- The menu has information on TGC, the developers, the meaning of the game, and "How can I make a real difference?" in terms of the water crisis.

- Everything in the game is in Grayscale except for the main character and the water droplets. The reason for this is because the clients wanted the main character and the water droplets to be the focus of the game.
- There is a comic book story introduction.
- The protagonist is an African-American girl with natural hair.
- Level 1 background is Flint, level 2 background is Pittsburgh, level 3 background is the whole United States.
- A skyline background must be used for levels 1 and 2, and level 3 background will have corn fields, mountains, and trees to represent the whole United States.
- For level 1 the character's weapon is an umbrella, for level 2 the character's weapon is a book and a pen, and for level 3 the character's weapon is a comb.
- The character's hair is braided for the first level, puffs for the second level, and finally an afro for the third level.

System Architecture

System Breakdown

Based on our functional and nonfunctional requirements, we designed the following system architecture as shown in the following UML diagram (Figure 1).

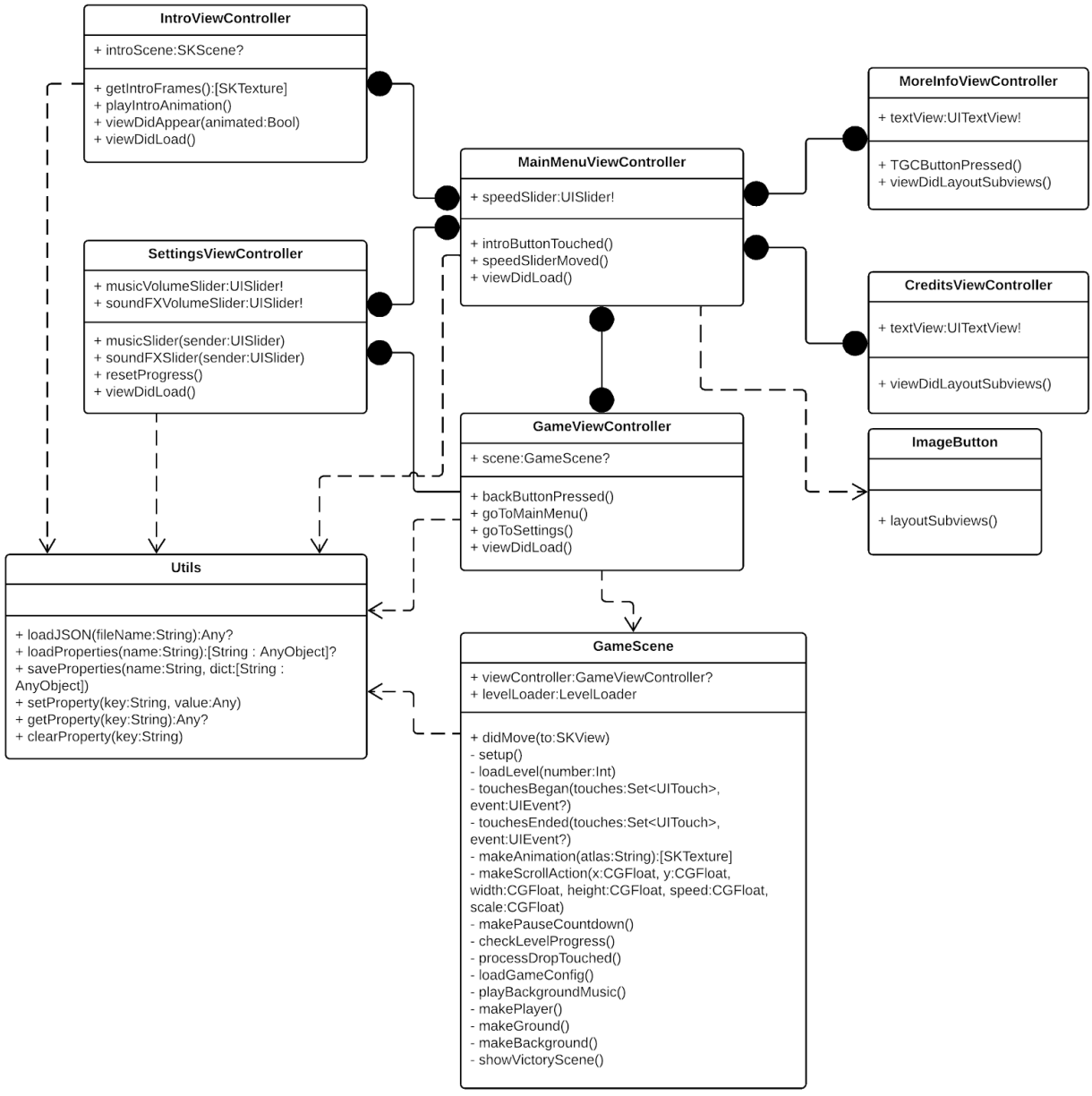


Figure 1 - User Interface UML

The responsibilities of the main components of our system architecture are described below.

- The **ViewController** classes are responsible for representing the user interface content of the application. Navigation between these view controllers is controlled through segues, which can be triggered by buttons or programmatically through a triggered event. Each view controller corresponds to a particular “screen” in the application:

    - **IntroViewController** - displays the comic book introduction sequence to the game
    - **MainMenuViewController** - the central hub of the application, contains buttons leading to gameplay, settings, credits, more information, and replaying the introduction
    - **SettingsViewController** - the settings page for controlling in-game volume and resetting progress
    - **GameViewController** - a container for the *GameScene* instance, responsible for connecting the gameplay to the rest of the application user interface.
    - **CreditsViewController** - displays the credits page
    - **MoreInfoViewController** - displays more information about the water crisis, contains a button linking to The Giving Child's website
- The **GameScene** is the main functional component responsible for pulling configuration files, loading assets, and running the main gameplay. This scene is presented through a *GameViewController*. *GameScene* is described further in the technical design section.

Storyboard

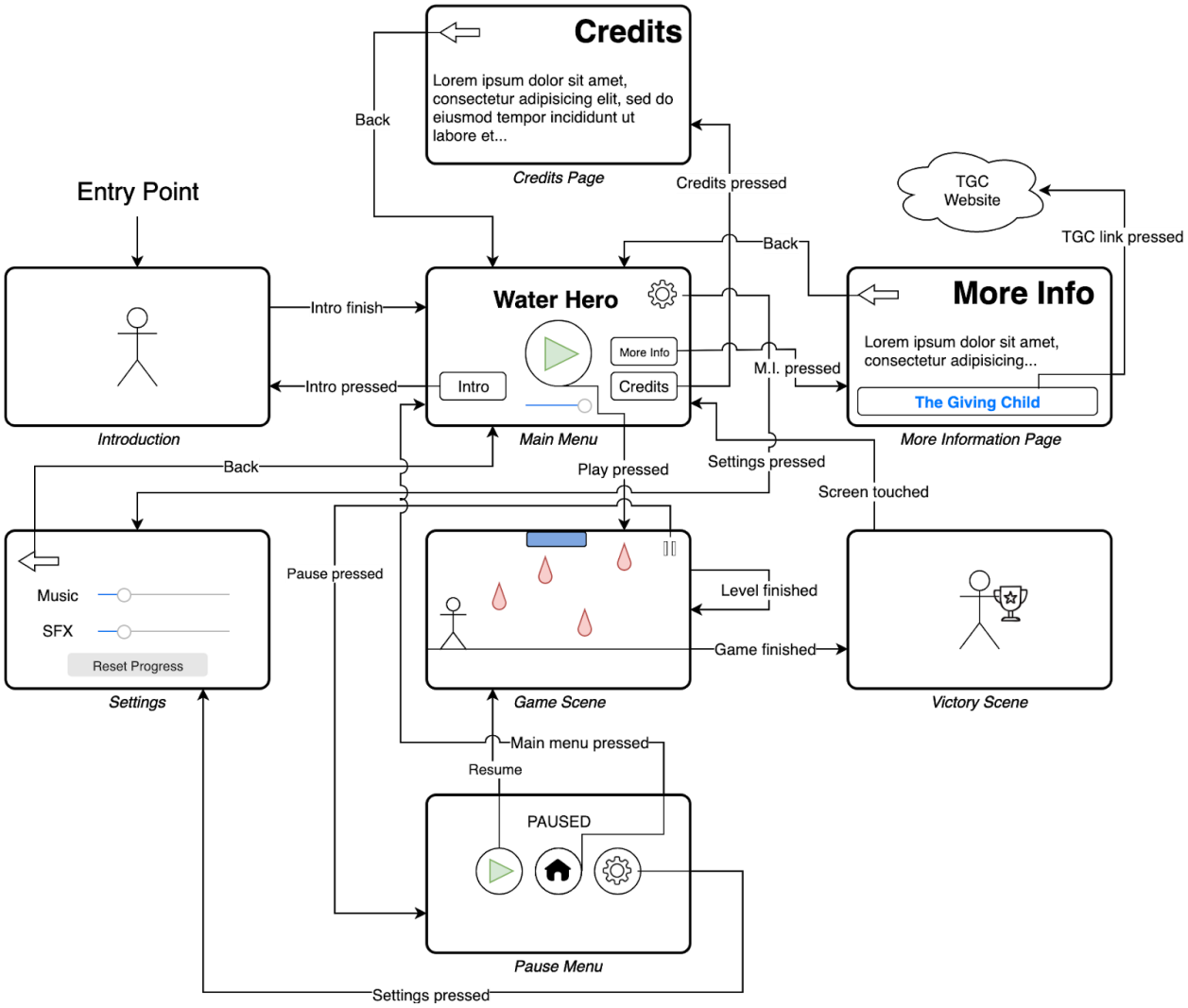


Figure 2 - Storyboard

The storyboard (Figure 2) shows the navigation between the different pages of the application. For example, when the app is opened, the user is shown the introduction and then taken to the main menu. Then the user can tap on the settings icon on the main menu which takes them to the settings page. Once the user is done adjusting their settings, he/she can tap on the back button to return to the main menu. The relationships between the other pages follow a similar structure whose transitions can be seen in the figure as well.

Technical Design

The brain of our application is the *GameScene* class (Figure 3). This class is responsible for handling the graphical user interface of the game itself. It also creates the interaction between the user and the game. In addition, it has to take into account the normal gameplay and the input from the player. *GameScene* makes the experience of playing Water Hero a seamless transition between human and computer. In the rest of this section, we will outline the tasks set upon the different parts of this class.

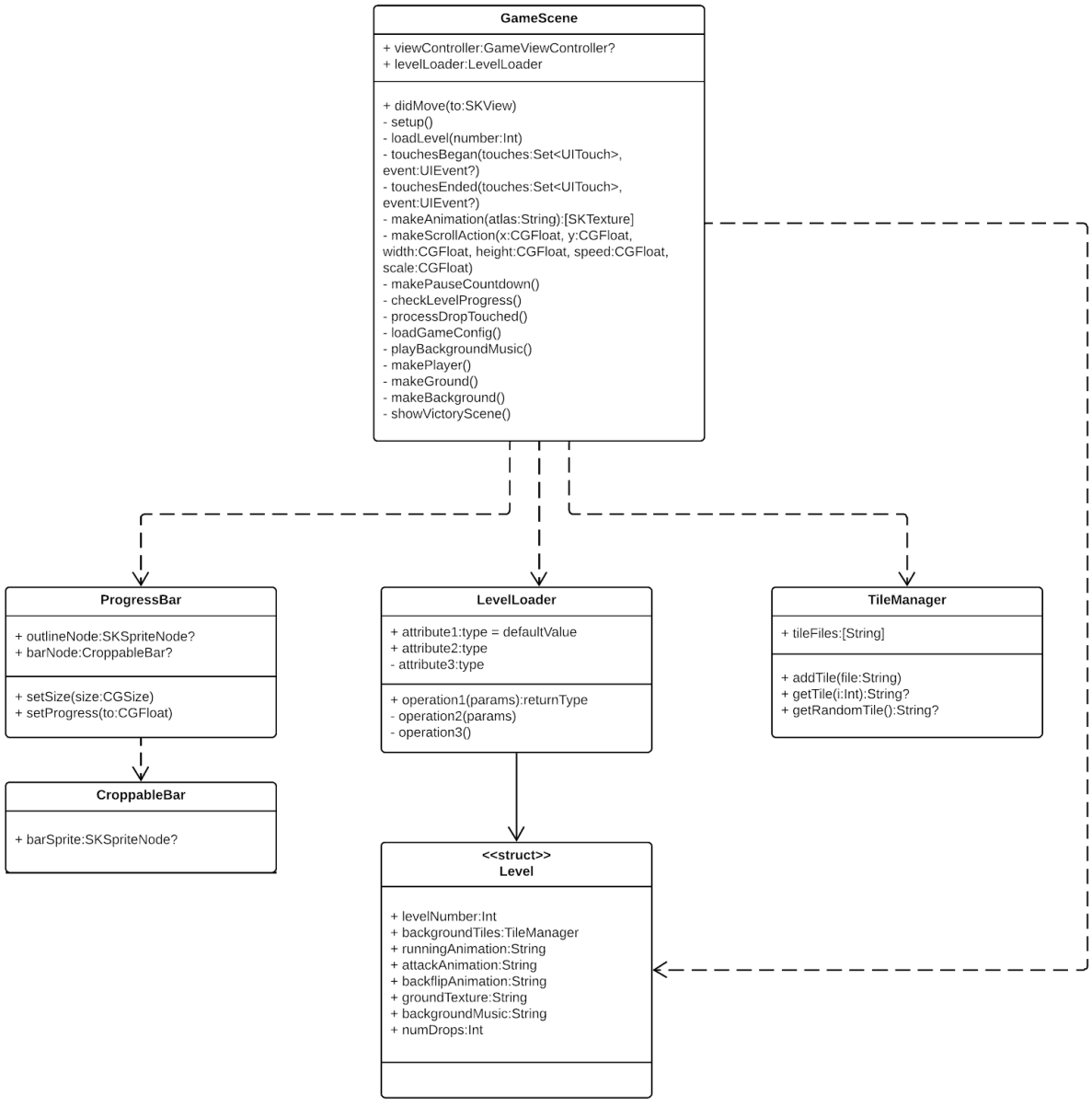


Figure 3 - GameScene UML

GameScene

Auxiliary Classes

Here at the Colorado School of Mines, we learned to write quality code and maintain good coding practices. These practices include splitting the functionality of a project into as many classes and files that are needed. For the Water Hero game, we divided the application up, using one main class to link everything together. The “glue” for our application is the *GameScene* class, but before we cover that, we will expand on our additional functional classes and structs.

The only structure in our application is the *Level* struct, which is responsible for modeling the current level. Using a structure to encapsulate the pertinent level assets (e.g. background images, music, animations, etc.) and the properties associated with the level (e.g. the number of drops popped for completion) is preferable since we do not need the additional functionality of a class.

For the rest of the application, we placed the remaining functionality in classes. In *GameScene*, we have *CroppableBar*, *ProgressBar*, *LevelLoader*, and *TileManager*. These components work together to add functionality to our GameScene class.

First off, our *CroppableBar* class is only responsible for creating the progress bar image that is used in the game. The *CroppableBar* class declares an *SKSpriteNode* and a *CroppableBar* to relay the current progress to the user. To set the size of the bar and actually send the current value for the progress, the *CroppableBar* class uses a *setSize* and *setProgress* function.

Supporting *CroppableBar*, our *ProgressBar* class handles the logic for the progress bar. It reads in the number of drops the user has tapped on from the game scene object and fills up the progress bar relative to the number of drops needed to move on to the next level. For example, if the user has tapped 75 water drops on level one, then the progress bar would be 50% filled.

Next, LevelLoader primarily stores information for the level using a Level array. This class also reads from a configuration file the file names for other JSON files, respective to individual levels. These other JSON files contain the basic information needed to create the level, in addition to features specific to the current level. This information is initialized to the Level array and thus the level can be successfully loaded.

Finally, our *TileManager* class primarily stores information for the background design file name in an array of strings. *TileManager* is responsible for adding, and accessing tiles (file names for the background). It is also responsible for randomly selecting the tile that is going to be displayed.

Through the distribution of functionality, we have kept the Water Hero application easy to extend upon in the future. At the same time, the user experience is seamless and enjoyable.

Human Interaction

At the beginning of computing history, machines were often given instructions to carry out, which were completed without input mid-process. Sometime in the development of

computing, the idea of accepting user input during computation became a reality. Over the span of a few decades, the gaming industry was created using this concept at its core. For Water Hero to be a playable game, we must take the user inputs (touches on the screen) and decipher their meaning in order to update the game accordingly.

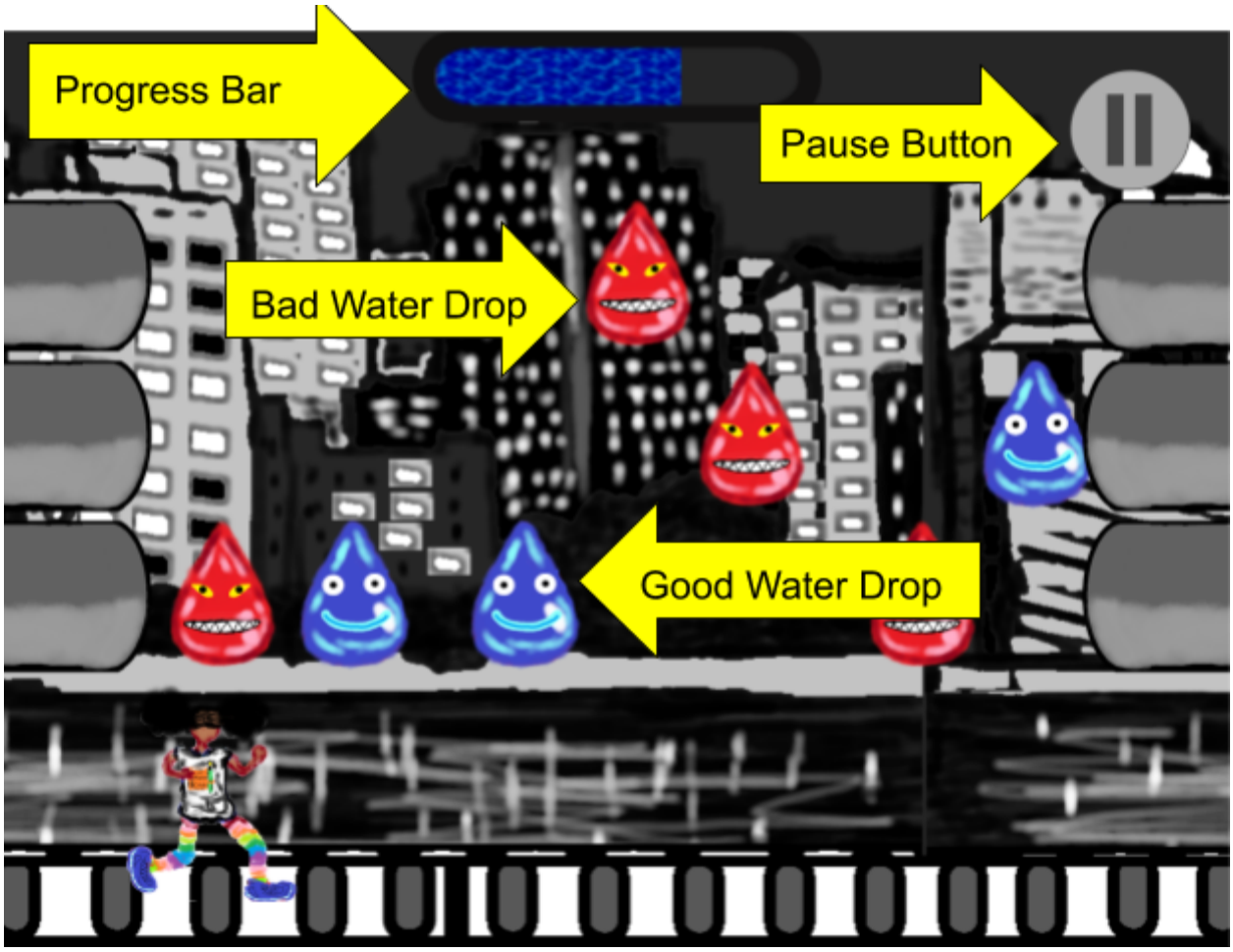


Figure 4 - Gameplay

The input is completely touch-based. This means that in order to allow the user to input commands, we need to display images as buttons and assign a backend to each of these buttons. Figure 4 is a screenshot of the game in action and it shows the pause button, the good water droplets, the bad water droplets, and the progress bar. The pause button and the water droplets are examples of buttons that the user can tap on.

In order to create the game, the *GameScene* class has a variable that is a *LevelLoader* object. Then, the *LevelLoader* reads from a JSON file and uses that configuration to create a *Level* struct. Next, that struct is passed into *GameScene*. Now, the correct assets are loaded into the *TileManager* object within the *GameScene* class so the game knows which set of background tiles to use. Meanwhile, *GameScene* is always interacting with the *ProgressBar* class, continuously updating it to give the user real-time feedback.

Quality Assurance

What We Did

In order to ensure that our product met the client's requirements and that the code is of high quality, we performed all of the following activities:

- Multiple playthroughs of the game to find any bugs and to test some of the required functionality. This activity checked for the performance of the application and the absence of defects.
- Unit testing to make sure that all of the functions work properly. This checked for maintainability and craftsmanship. Listed below are the things we wanted to test:
    - Reading/writing configuration files to make sure levels are loaded with the correct assets. These assets include the set of background images the background tile manager will use, the character model to use, the background music file to use, the ground image to use, and the number of drops needed to proceed to the next level.
    - Loading random background images using the Background tile manager. This was done by choosing a random image from an array of images (which is loaded from the configuration files) to use as the background at a particular moment in the game. Basically, the background is a loop of images and we wanted those images to be randomly chosen as the game progresses.
- User interface testing to make sure that the user has an enjoyable and comfortable experience with the application. This also checked for maintainability and craftsmanship, but in a separate aspect of the project than the unit testing. Listed below are the aspects of the user interface that we tested:
    - Settings button on the main menu takes the user to the settings page which has music volume, sound effects volume, and an option to reset the game's progress
    - More info button takes the user to the more information page which has information on the water crisis and The Giving Child
    - Credits button takes the user to the credits page which has a list of everyone involved with the project
    - Intro button replays the intro of the game
- Having separate configuration files for each level of the game so that the settings for each level can easily be modified by either us or future developers. This accounts for maintainability and craftsmanship.
- Static program analysis to check for craftsmanship. The program we used was CODEBEAT which connects to our private Github repository and automatically runs an analysis each time code is pushed.
- Multiple scrum sessions throughout the project. This was done so everyone was on the same page concerning the tasks that were going to be accomplished for the day.

Addressing Risks

There were three technical risks that we were accounting for:

- The first was if the game had any bugs or glitches. In order to make sure that there were no defects in the game, each one of us played the game (in the same manner that a child would) to look for anything that does not work as intended. For instance, some of us tapped on the screen wildly in order to make sure that the program could handle a large amount of input in a short time frame. Some of us even played the game incorrectly (i.e. only tapped on the blue water drops) for about five minutes to see if that would break the game.
- The second technical risk was that the game fails on some Apple devices while it works on others. Among the team there were a couple of iPhones and iPads that we used to test the game on, so that was how we addressed this risk. We have confirmed that the application works on both iPhones and iPads.
- The third risk is that the game got rejected by the App Store. Since we want this game to be played by children all across the United States, we did not want the game to be rejected. In order to deal with this potential problem, we planned to finish the game early and submit it as soon as possible so that if it was rejected, we would have time to fix the cause for the rejection and submit the game again.

Validation

The team verified that it met all of the client's requirements by showing the client a finished version of the game. Leading up to the final version, we had been showing a demo of the game to the client during our weekly meetings. This was done so that they could provide feedback on the requirements we had met so far and if it was up to their standards.

Results

Unit Testing

The unit testing results for each aspect that were mentioned previously were as follows:

- Reading/Writing configuration files.
    - We have a unit test for each level configuration file that makes sure that each one is read correctly by the game scene class. What we are checking for is that the correct variables are used in each level. For example, we have three different character designs, each corresponding to a different level. We want to make sure that the correct design is used in each level. Other things that the tests check for include: the set of background tiles that should be used, the animations that should be used, the ground texture that should be used, the background music that should be playing, and the number of drops required to proceed to the next level. All three of these unit tests passed.
- Random Background testing.
    - Another thing that we have a unit test for is the get random tile function in the tile manager class. In order to test for randomness, we ran the function 100 times on an example set of tiles and kept track of how many times each one was chosen. We considered the test sufficient if each tile was chosen at least once out of the 100 iterations. This test passed.

User Interface Testing

Our User Interface Testing results for each aspect of the User Interface tests we discussed previously are as follows:

- Settings button on the main menu successfully takes the user to the settings page.
- More info button now takes the user to the more information page which has information on the water crisis and The Giving Child. There is a Giving Child button link which opens safari linking to their donate page on their website
- Credits button successfully takes the user to the credits page which has a list of everyone involved with the project
- Intro button successfully replays the intro of the game

CODEBEAT Results

Lastly, we obtained our final results by using CODEBEAT for static program analysis. A sample picture for the CODEBEAT analysis is shown below (Figure 5).

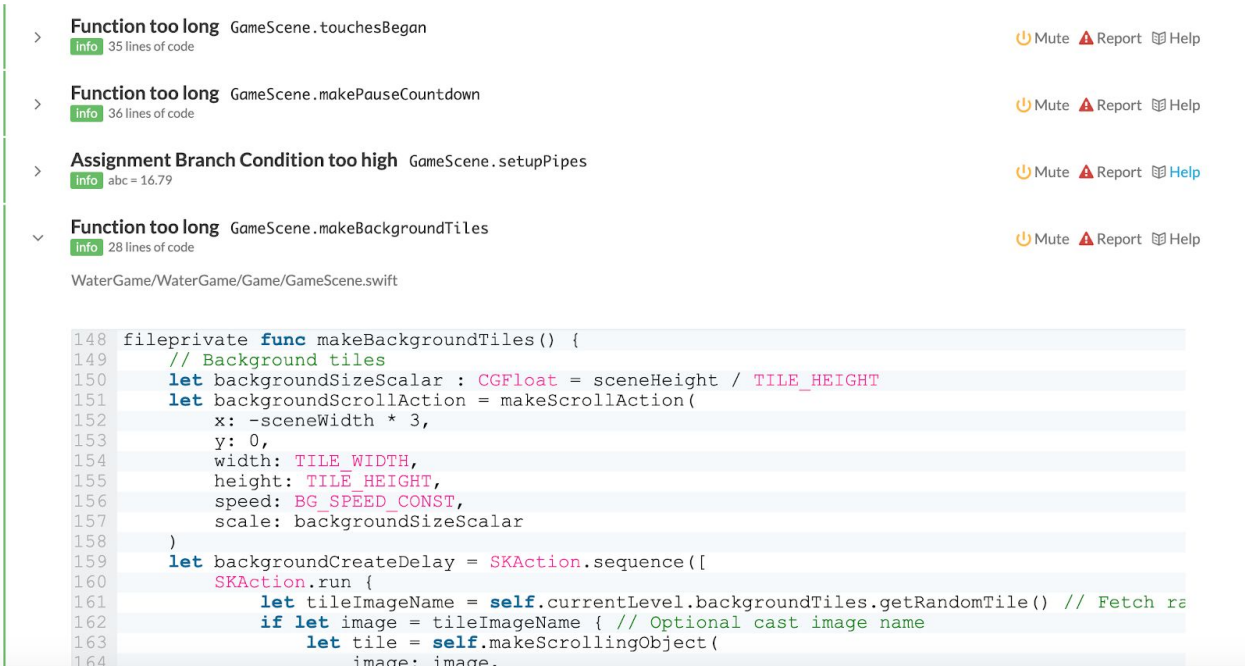


Figure 5 - CODEBEAT Analysis

Some of the analysis CODEBEAT gave us was understandable; however, we thought some were unnecessary. Therefore, we did not make changes to some parts because we thought the code was still maintainable. For example, a common complaint that we saw was that a function was too long, even though it has less than 100 lines of code. We believe this analysis is purely subjective and thought it would be unreasonable to refactor these functions since they work perfectly fine while maintaining high readability.

To summarize, all of our unit and user interface tests have passed. We believe that the tests we have written in conjunction with actual playtesting are sufficient for ensuring a bug-free product.

Lessons Learned

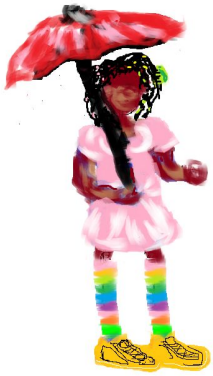
- This game was made entirely with Swift Programming Language. Before starting this project, none of us had any experience with Swift. Now, we have learned the basics of the language and can make our own iOS app/games in the future.
- We've usually done coding individually or in pairs for homework and projects. Now, we've gained some experience in working on a larger project with more people involved. While working on projects in the real world is a lot more complicated and usually involves a lot more people, this project gave us a taste of what it's like working in a group on a larger project.
- We have learned what it's like to work for an actual client. The process of creating a product that should satisfy the client involves a different kind of stress than simply doing an assignment for class. This stress was brought about by the fact that field session - in addition to being a class - can count for real work experience, and we all wanted to get the most out of the six week period. In the end, it was this stress that made us produce high-quality work by making us more focused and helped us all grow as software engineers.

- Few of us have worked in an Agile environment before, so working with Scrum was a great learning and experimentation process. In addition to the particular tools that we used to maintain our Scrum process, we have learned a great deal about complexity estimation, resolving issues early in the development process, scheduling self-reflection, and being flexible enough to stay on top of deadlines while maintaining stability.

Appendix

- Graphics were done by Abhaya Shrestha, Ryan West, and Joseph Thurston. Some sample graphics are shown below created based on our client's non-functional requirements.

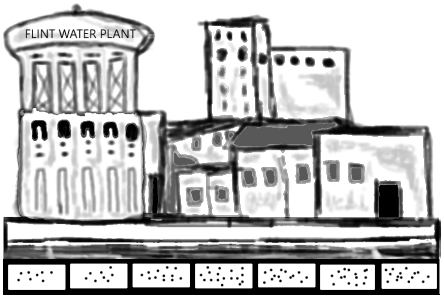
Sample Character Design for Level 1



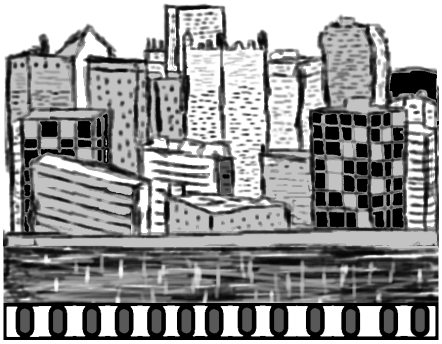
Sample Character Design for Level 2

Sample Character Design for Level 3

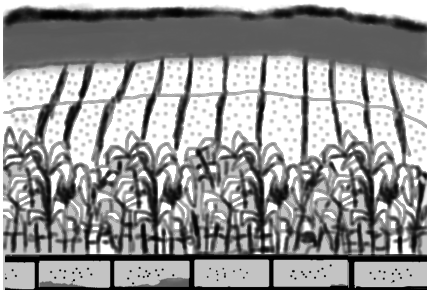
Sample Background Design for Level 1



Sample Background Design for Level 2



Sample Background Design for Level 3



- The application will be available for download on the Apple App Store.
- User Acceptance Testing with the character design was done by Evie Barnes from The Giving Child Organization as follows:
    - Evie showed the design to a group of women of various ethnicities and got their feedback.
    - Their feedback was forwarded to us in order to adjust the designs.
    - Overall, the feedback was positive, and few modifications needed to be made.
- All rights to the game belong to The Giving Child Organization.

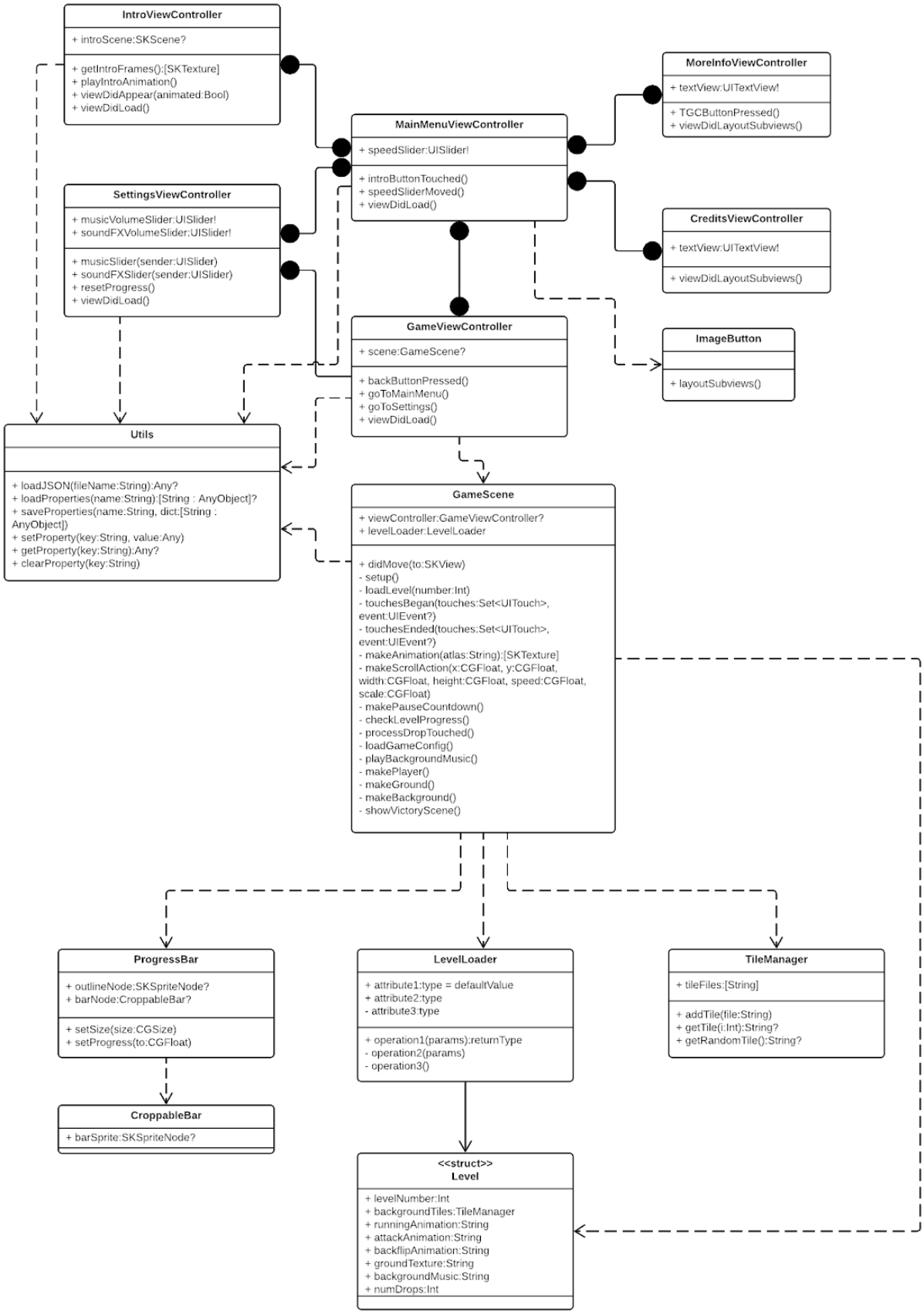


Figure 6 - Full UML