Combined Design Document

for

Duck Tower Defense

Cody Chu, William Robichaux, Ezekiel Velasquez, Lucas Riccitelli, Brian Larosiliere, Emory Salaun, Gabriel Badolato

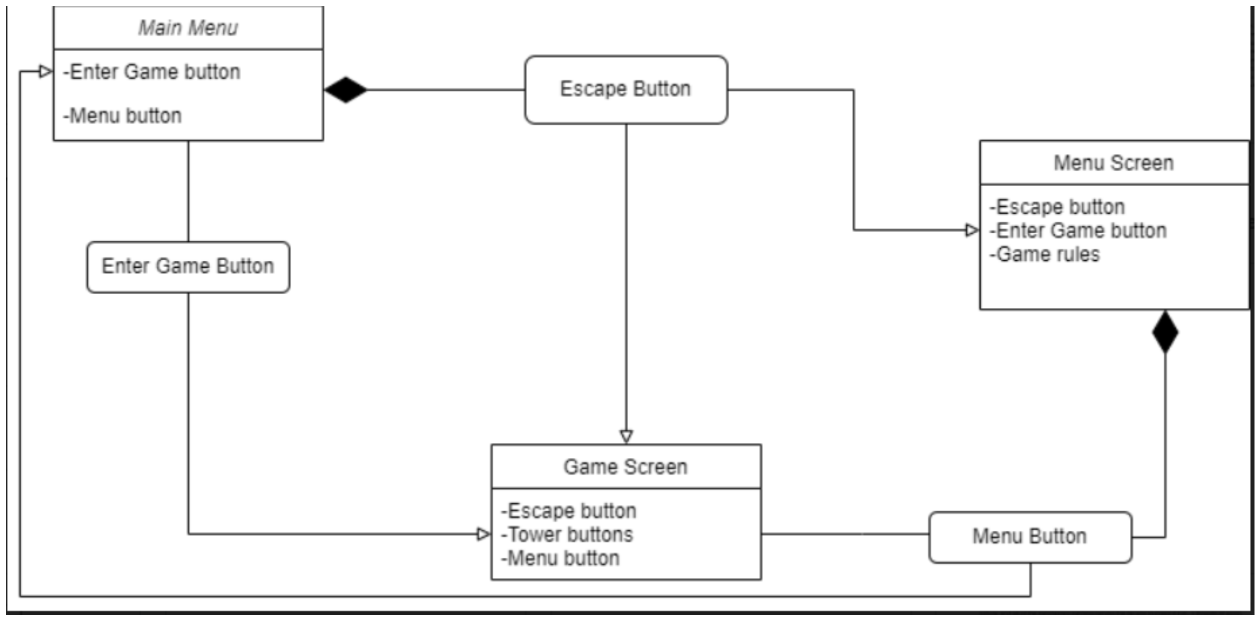
Team K

11/30/2022

Table of Contents

3.1 Main Menu

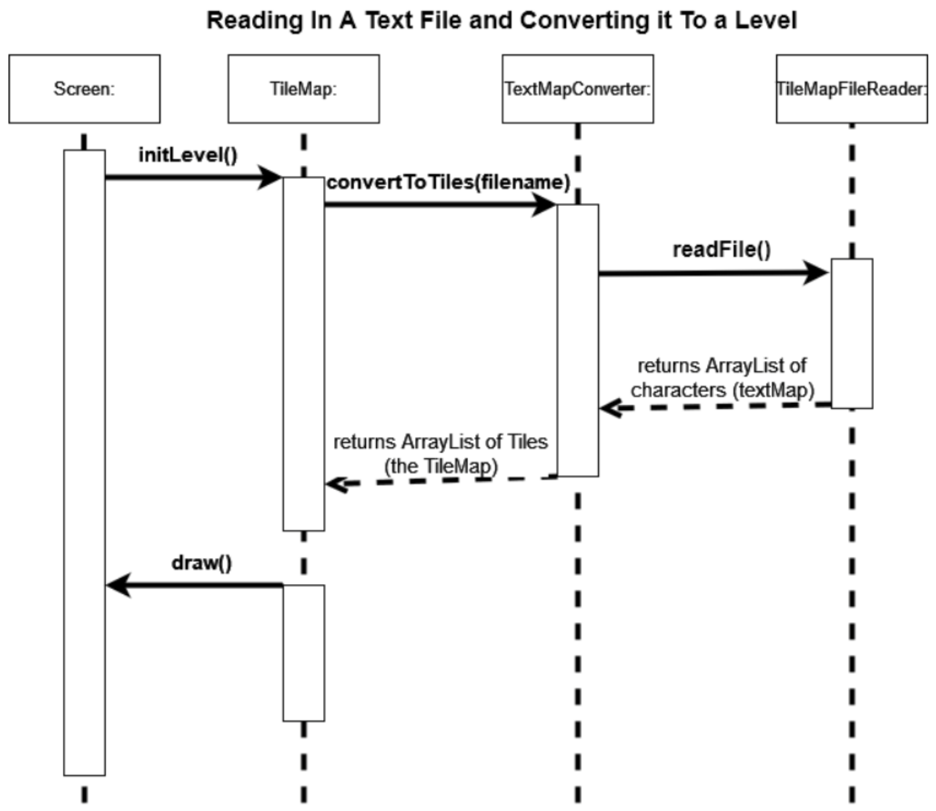
I chose to do a UML class diagram for the design document because I felt it would fit with the class that I am describing. What I have drawn is how the screen classes interact with each other through the buttons. From the diagram It starts with the main screen that pops up and two options are given either to go to the game screen or the menu screen. From either screen you are able to return to the main menu through the escape button. When you are on the game screen you are also able to go back to the main menu or the menu screen depending on which button is chosen. What I was able to see through this diagram is that all of the screens work within each other meaning that no matter what screen you are currently on you can travel to any of the screens that are desired.



3.2 Tile Map

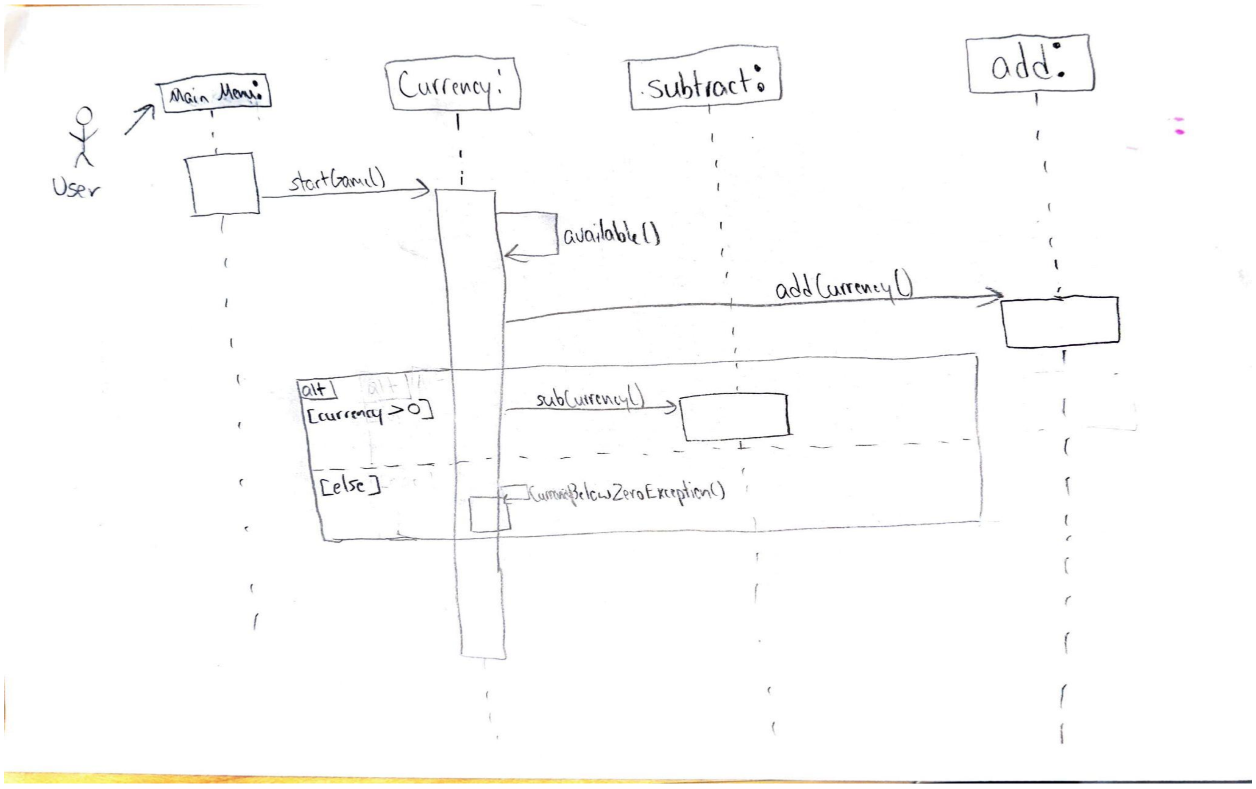
Emory Salaun
The piece of our software that I decided to make my individual design contribution was the system used in reading a text file, converting it into a usable format, and then being able to display that to the screen as a level. In order to show this clearly, I have created a sequence diagram of how it works below. We had 3 functional requirements for this part of our project: to convert a text file into an array of characters, turn those characters into usable tiles, and then display those tiles on the screen. The system is originally called by the Screen when first loaded, and the first function called is to initialize the level (initLevel()). This TileMap method

then calls TextMapConverter’s abstract function convertToTiles(), which takes an ArrayList of characters, converts them into tiles based on their character, and then returns the list of Tiles to the tileMap. The TextMapFileReader provides the ArrayList of characters to be converted after the readFile(filename) method is called. Overall, this system provides an easy way to create levels as a text file using ASCII symbols, and then can initialize these text files as actual levels to be used in the game


Reading In A Text File and Converting it To a Level
Screen:
TileMap:
TextMapConverter:
TileMapFileReader:
initLevel()
convertToTiles(filename)
readFile()
returns ArrayList of characters (textMap)
returns ArrayList of Tiles (the TileMap)
draw()

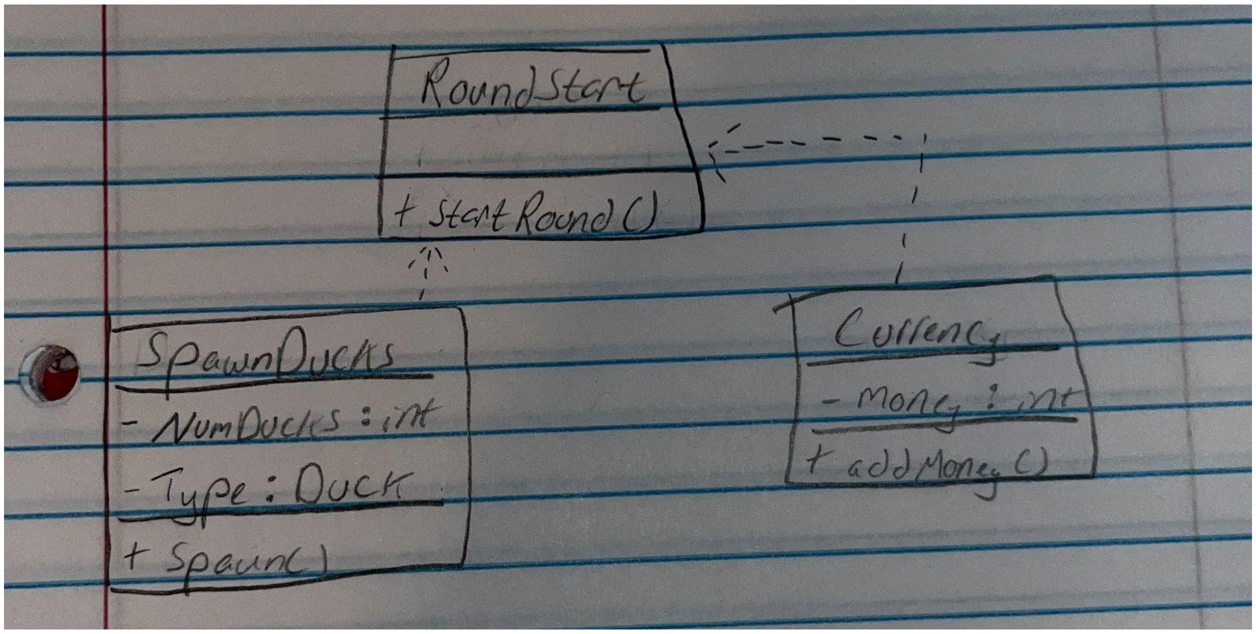
3.3 Currency

For my UML diagram contribution, I created a simple sequence diagram for the Currency feature. Although it is a basic class, this diagram explains the sequence of steps the program will take right from the very beginning. With that being said, I included a user beginning from the Main Menu feature of our program in order to help showcase what I mean. I included this Main Menu feature into my diagram because it is needed to begin the game in which the Currency feature will begin to take effect throughout the entirety of the game. After starting the game from the main menu, Currency is able to self-call in order to show the user their available funds, which are set with the class itself. From there, the currency class has two other main sequences to perform, those being subtract and add. In order to subtract, the user's currency value must be over 0 since one's funds cannot be negative as well as doing a function in the game. On the other hand, in order to add, another function must happen in the game. For the subtraction of currency to happen, the user must first place down a tower but because this is from another class, I simply put subCurrency. This same thing goes for the add method since the addition of currency comes from the "destruction" of ducks, which is another class, hence why I simply put addCurrency. If all of these conditions are met, then the Currency class is able to work in this sequence.



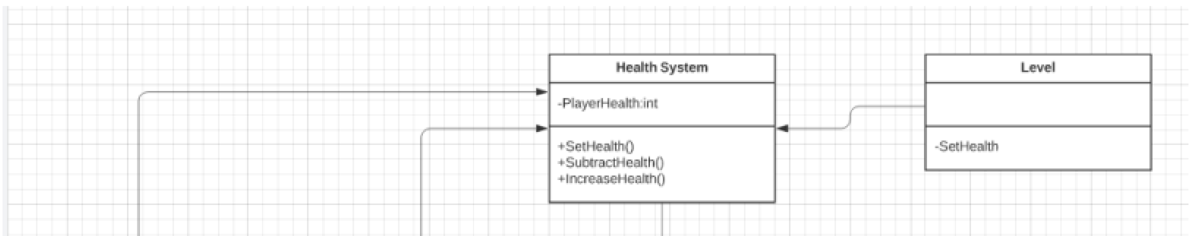
3.4 Round Start Button

The class that I created my UML class diagram for is the roundStart class(3.4). This class takes care of what happens when the user presses the start button in between rounds of the game. The two main things that happen when a round is started in the game, a predetermined amount of ducks spawn(3.4.3.4). Depending on the round, the ducks will be of a different type. This can mean that the ducks have more health, which would increase the amount of damage the player takes, or how fast they are. The other thing that happens during the start of a round is the player will automatically receive a set amount of money which scales with how far through the game the player has made it(3.4.3.3).



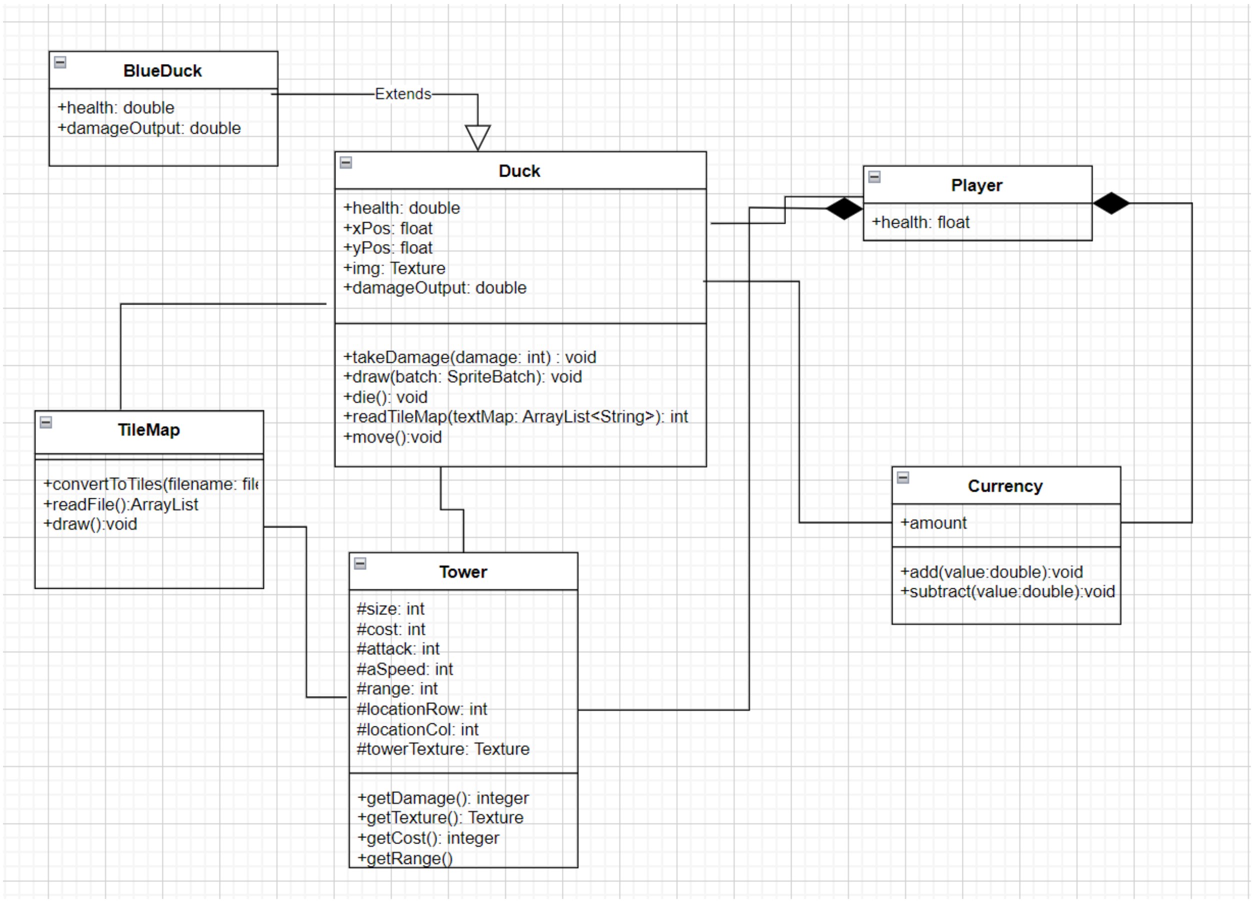
3.5 Health

For my UML Diagram contribution, I created a diagram for Health System feature (3.5). This diagram explains the sequence of steps required for the health system in our game. The health system functionality is to represent the user health during the duration of the game. 3.5.1 is the health system class as the player starts of with an integer as the health. 3.5.2.1 is represented as the enemy class. As the enemy duck crosses a certain boundary the health bar should be subtracted. 3.5.2.2 is implemented as healing. As a certain requirement is achieved for healing, health is added and updated to the health bar. 3.5.2.3 is represented to set the health per level.



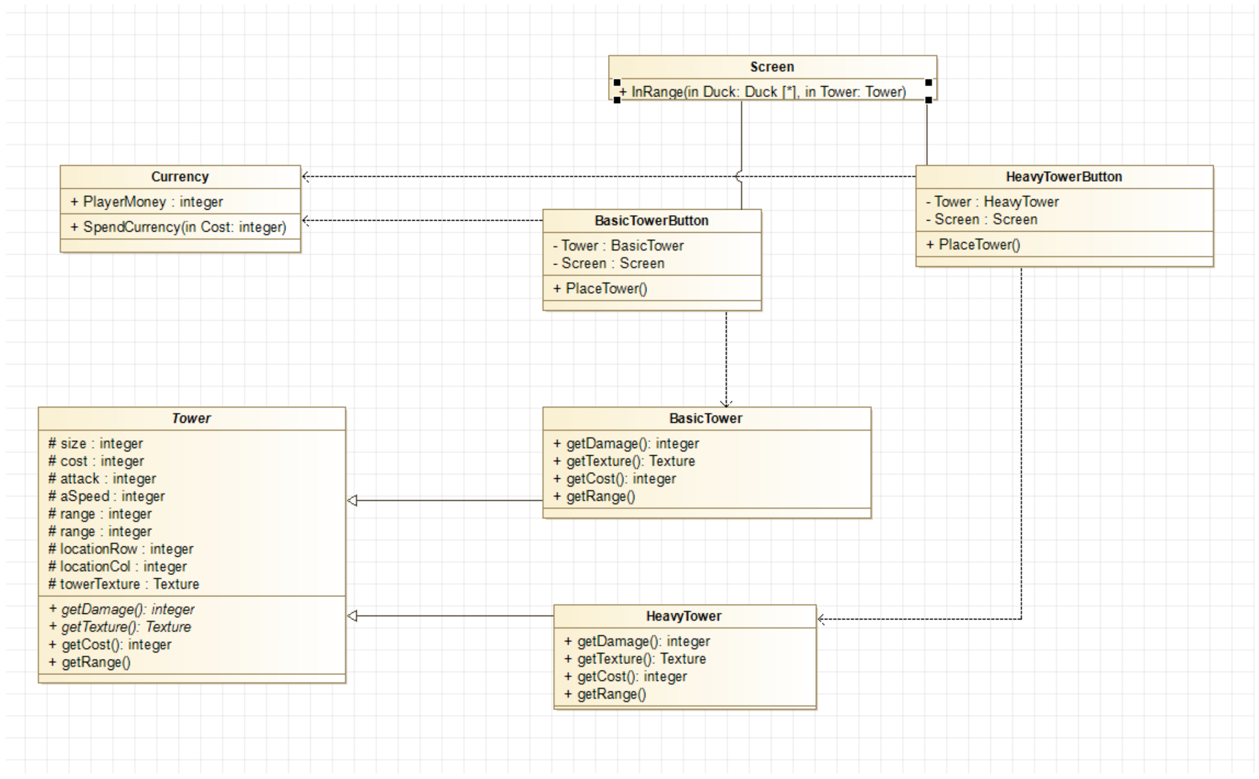
GUI is implemented in this diagram because health determines if the user advances to the next level, wins the game, or loses the game.

3.6 Ducks



For my UML diagram contribution, I made a class diagram of the Duck Class. This diagram describes the sequences of steps of spawning and destroying ducks. As described in 3.6.2.1, the duck is associated with the player as the duck is capable of doing damage to the player's health. The association between the duck and tower is described in 3.65.2. The association between the duck and tilemap is described in 3.65.4. At 3.3.2 we can see the association between the tower, tilemap, and currency. We can also see the association between ducks, currency, and player at 3.4.2.

3.8 Towers

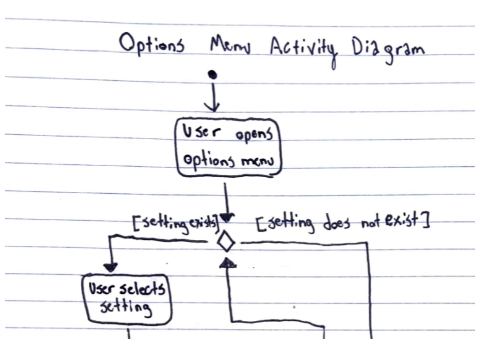


In order to represent the tower classes (SRS 3.8) I drew a class diagram. This diagram is used to show the hierarchy of the tower classes and their interaction with other classes in the program. The *Tower* Class is an abstract class that is used as a factory class that built the BasicTower class and HeavyTower class. The corresponding Button classes are used to interact with the screen class by giving the User a way to purchase and place towers. The Button classes call for their corresponding Tower object to be made when the user clicks on the button and has enough currency to purchase the tower. The currency is then subtracted from the currency class and the Tower object is added to the Screen, which fulfills feature 3.8.3.4. Note: Currency and

Screen have a few more classes that are involved in them but I bundled them into two minimalistic classes for readability. The Screen handels the location of Duck and Towers objects and will call the InRange() method to check all Duck Objects on the screen and targets the duck furthest along the track, within range. If there are no Ducks in range it does nothing. The method then applies damage based on the getDamage method from the Tower class. Which fulfills features 3.8.3.1, 3.8.3.2, and 3.8.3.3.

3.9 Options Menu

For my document contribution, I worked on the options menu for the game (Section 3.9 in the SRS document). The goal of our project is to create a two-dimensional tower defense game. My contribution to this project is the creation of the options menu. I thought it best to represent this addition in the form of a UML activity diagram. The UML diagram depicts the decision-making process that the user goes through once this menu has been opened. When they open the menu, they must decide whether the setting they want to modify exists. If the desired setting does not exist, the user then closes the menu, otherwise, they select the setting that they wish to modify. Once modified, the user decides whether or not they are satisfied with the current settings. If the user is, they then close the options menu, otherwise, they return to the first decision point, and repeat this whole process. I created this UML diagram to represent the options menu as it exists now, we may add more settings in the future, which would add to the decision points that were described above, and shown below.


Options Menu Activity Diagram
User opens options menu
[setting exists]
[setting does not exist]
User selects setting