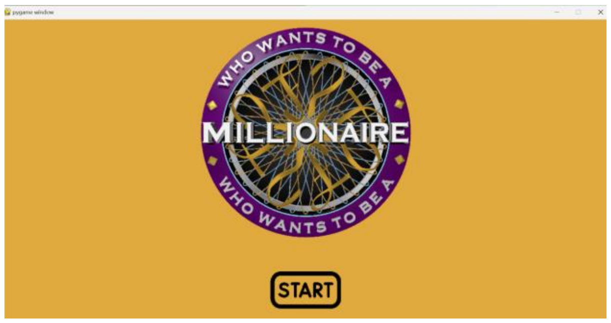
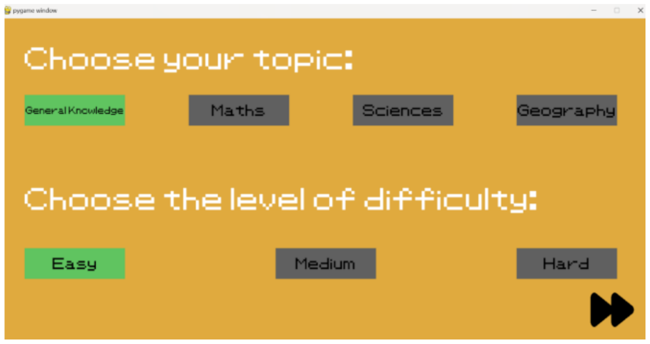
AIST1110 Object Oriented Programming (OOP) Report

Rules and Game Flow

1. On the action of **running the main_game.py module**, player will be shown the screen "Who Wants to be A Millionaire"

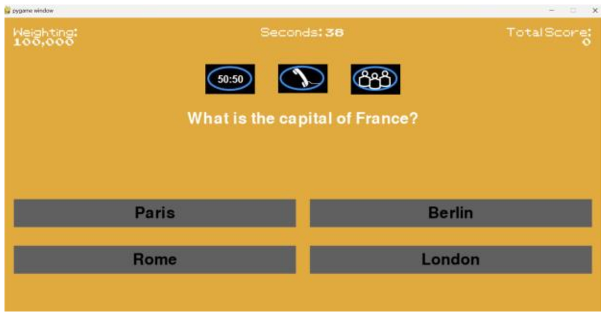


2. After clicking the start button, player will be redirected to the "Selection Screen" wherein player will have to choose the topic for the questions and the level of difficulty of the game. The radio button (for the option) is by default set to "General Knowledge" and "Easy". Easy mode is around junior high school level, medium mode is similar to senior high school level, and hard mode is nearly the university level. However, the harder the question is, the more likely the API hallucinate, failing to generate the questions. Testing the game using "General Knowledge" and "Easy" Mode will guarantee the generation of questions.

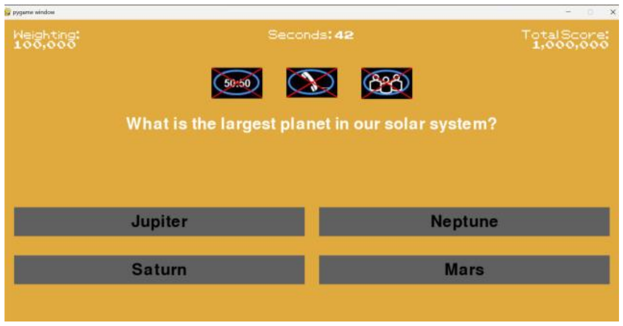


3. By clicking the black arrow button, the connection with the chatgpt API is established and the input from the user selection will be used by the API to generate the questions, choices, as well as their corresponding answers. Please note that the arrow button clicked will trigger the connection with the API hence it will take a longer time to respond and to redirect to the next screen.
   Note: We actually wanted to generate 20 questions, however the API keeps on hallucinating and crashing for bigger datas. Hence, the maximum number of questions we can generate is 15.

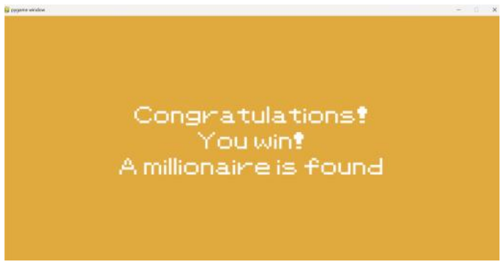
4. After the connection is established, the display is as follows. With in total 15 questions, each weighting 100,000



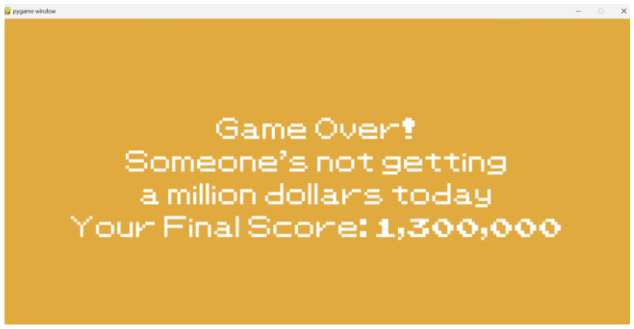
After each lifeline is clicked, it will cross the lifeline (indicating only once usage). The fifty-fifity will remove 2 answer options, while the phone a friend will show the answer in the format “Your friend said: ANSWER” and ask the audience will show the answer in the format “The audience said: ANSWER”. If player attempted the question correctly, the weighting will be added in the total score.



5. If the player was able to attempt all the 15 questions with or without lifelines, the game will output the following screen and play the clapping sound


Congratulations!
You win!
A millionaire is found

Else, if the answer is wrong or player ran out of time (which was set to 45 seconds), it will output the following screen as well as the losing sound effect.


Game Over!
Someone's not getting
a million dollars today
Your Final Score: 1,300,000

Limitations and Improvements

This game is heavily dependant on chatgpt API's performance. Once the API failed to generate the questions, the user will take even longer to wait for the question screen being displayed. A recommendation to fix this issue is to have a series of backup questions for each category that can be called once the program detected that the API failed to generate the questions according to the desired format.

Analysis of OOP Usage

In terms of our OO design, it has a total of five modules: screens, questions, widgets, game and main_game.

1. Screens Module

In this module, we are trying to separate different screens into different classes to ease the tracking of errors as well as the flow of the game.

1. 1. Class Resource:

   Contains all the resources such as screen width, screen height, background music, clock, sound effects, fonts, and buttons. This is to ease the usage throughout all the functions and modules in the future.

__enter(self):
This function is to enter the main game and utilizing the resources

__exit__(self):
This function is to exit the main game

use_callafriend(self):
This function is to initialize the lifeline call a friend

use_eliminate50(self):
This function is to initialize the lifeline fifty-fifty

use_asktheaudience(self):
This function is to initialize the lifeline ask the audience

1. 2. Class MultilineText:

   This class functions to regulate the display of the questions into multiple lines in case the length of the sentence or question is longer than the width of the screen. Adjusting the proper amount of spacing and new lines so that even the long questions are readable to the player.

   __split(Self, width):
   This function is to split the lines when the width of the lines are longer than the width of the screen. It uses loops and checking to get the divisions and segments. It also utilizes math.ceil by importing the math module

   get_height(self):
   This function is to get the height of the line after each splitting

   __display_multiline(self):
   This function serves as the private function that hides the specific details about displaying the multiple lines on the question screen. It eases and simplifies the content presented in the display() function as it is then called in the display (self) function

   display(self):
   This function calls the __display_multiline() function, __split(), and the get_width() functions as it acts as the mother functions which outputs the display onto the screen

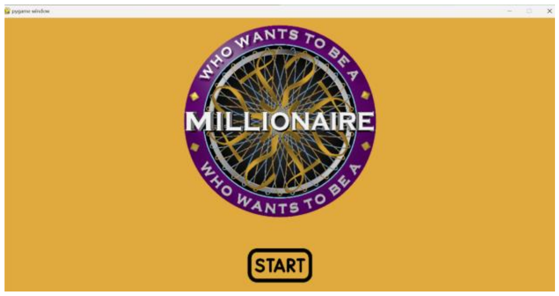
1. 3. Class Screen(ABC):

This class utilizes the abstract class and abstract method to further emphasize on inheritance, polymorphism, abstraction, as well as encapsulation.

display():
This is the method that will be overwritten by different screens in different stages of the game

1. 4. Class IntroScreen(Screen):

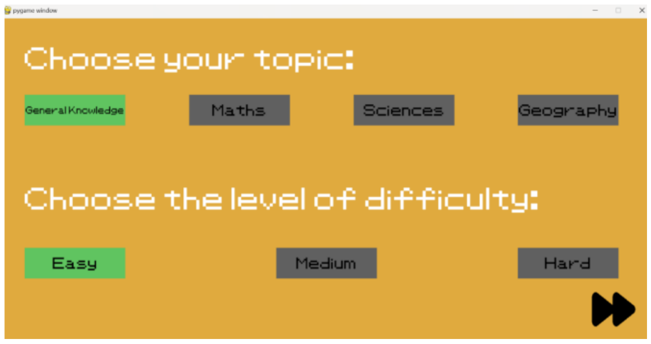
   This class inherits from the resource class to utilize the initialized fonts, texts, etc. and this class will overwrite the screen class. Below is the display of the intro screen



   display(self):
   This method overrides the parent class, screen(ABC)'s display() abstract method

1. 5. Class SelectionScreen(Screen):

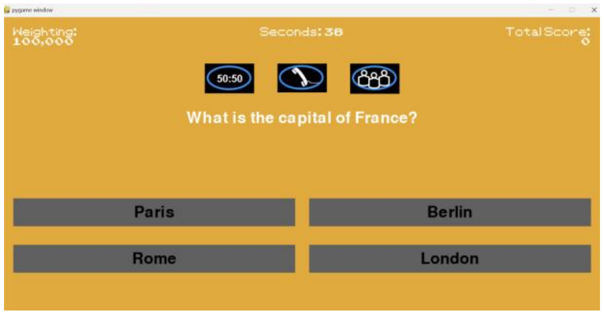
   This class inherits from the resource class to utilize the initialized fonts, texts, etc. and this class will overwrite the screen class. Below is the display of the selection screen



   display(self):
   This method overrides the parent class, screen(ABC)'s display() abstract method

1. 6. Class QuestionScreen(Screen):

This class inherits from the resource class to utilize the initialized fonts, texts, etc. and this class will overwrite the screen class. Below is the display of the question screen



is_timed_out(self):
This method checks if the player ran out of time (which was set to 45 seconds per question)

__show(self, event_list):
This is a private method, not accessible to other instances. This method shows the process of displaying the question screen. This method is later called by the display method in this class

display(self):
Display method will call the __show method to ease the displaying and overriding of the Screen class.

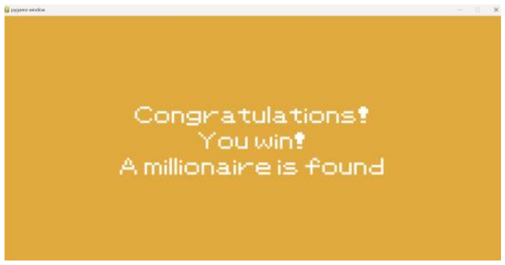
1. 7. Class MessageScreen(Screen):

   This class inherits from the resource class to utilize the initialized fonts, texts, etc. and this class will overwrite the screen class.

   display(self):
   This method is to format the message display of the screen (including padding and line separations)

1. 8. Class WinMessageScreen(MessageScreen):

   This class overrides the MessageScreen class to display the congratulatory message if the user wins

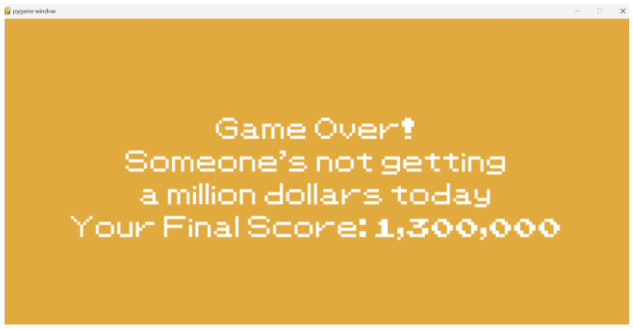


Congratulations!
You win!
A millionaire is found

display(self):
This method is to format the message display of the screen (including padding and line separations)

1. 9. Class LoseMessageScreen(MessageScreen):

   This class overrides the MessageScreen class to display the congratulatory message if the user loses



Game Over!
Someone's not getting
a million dollars today
Your Final Score: 1,300,000

display(self):
This method is to format the message display of the screen (including padding and line separations)

2. **Questions Module:**

This module is used to call chat GPT to provide suitable multiple-choice questions for the program and deal with the question and answer we get.

1

1.1 Class Qusetion:

This class is to ease the calling in other modules especially in the main game module. The main function is to fulfill the 50:50 role of randomly generating two wrong answers to eliminate and check the answer correct or not.

get_question_text(self):
What this function does is return self._question_text.

get_answers(self):
The function is to return self._answers.

get_correct_answer(self):
The function is to return self._correct_answer.

remove_two_incorrect(self):
The function is for the eliminate50 / fifty-fifty function in the main game. It randomly generates two items from the wrong answers and removes them from the choices.

check_answer(self, answer):
The function is to return true if the inputted answer is the same as proposed answer by chat GPT.

1.2 Class QuestionGenerator:

This class is calling chat GPT to give suitable questions with description, and check if chat GPT has generated answers based on our needs.

__get(self):
The function is to get the questions, choices, answers from chat GPT using Azure.

__next__(self):
The function implemented to pop and return the first question from a list of questions stored in the self. questions attribute. If the list is empty, it raises a StopIteration exception to indicate that there are no more items to iterate over.

__iter__(self):
The function simply returns the current object (self) as the iterator. This means that the object itself is an iterator, and it can be used directly in a for loop or with other iterator-related operations.

3.Widgets Module
This module aims to control the button in our game containing class RadioButton.

2
2.1 Class RadioButton (pygame.sprite.Sprite):

In this class, we must force the user to pick an option and that is visibly implemented in the selecting subject and difficulties (part of the code). It also eases the declaration of the radio buttons and the classifications of them.

Update(self, event_list):
The function is used to update the choice of buttons. For example, if we choose one of the answers of the questions, other button will update to be not chosen by the player.

setRadioButton(self, buttons):
The function is used to define self.button.

4. Game Module

In this module, we are trying to ease the pressure on main game. This module imports Question and QuestionGenerator class from question module.

3

3.1 Class Game:

The class is used to set the basic function of the game.

use_lifeline(self, lifeline_name: str, question: Question):
The function simulates the using of lifeline. If this lifeline hasn't been used, conduct it and remove it from the usable list referring to the Question class in questions module.

set_generator(self, generator: QuestionGenerator):
The function refers to the QuestionGenerator class in questions module to generate new questions.

add_score(self, points):
The function calculates the total score that the player wins.

__iter__(self):
The function simply returns the current object (self) as the iterator. This means that the object itself is an iterator, and it can be used directly in a for loop or with other iterator-related operations.

__next__(self):
The function is implemented to pop and return the first question from a list of questions stored in the self.questions attribute. If the list is empty, it raises a StopIteration exception.

3.2 Class GameQuestion:

This class is used to control the parts related to the answering questions.

get_score(self):
The function is used to calculate the score.

get_weighting(self):
The function is used to get the weighting of the question.

use_lifeline(self, name: str):
The function implements the flow of using lifeline referring to use_lifeline function in game class and qusetion module.

get_question_text(self):
The function is used to get the text of question.

get_answers(self):
The function is used to get the answer of question.

check_answer(self, answer):
The function is used to check if the answer provided by player is the same as the correct answer. If the answer is correct, the program refers to game class to add score.

3.3 Class Lifeline:

This class utilizes the abstract class and abstrant method to further emphasize on inheritance, polymorphism, and encapsulation.

use_lifeline(self, question):
This function works as an umbrella function for the lifelines.

3.4 Class AskTheAudience(Lifeline):

This class rewrites class Lifeline to specifically define the lifeline of ask the audience referring to question module.

use_lifeline(self, question):
This function returns the answers from audience referring to question module.

3.5 Class PhoneAFriend(Lifeline):

This class rewrites class Lifeline to define the lifeline "Phone a friend".

use_lifeline(self, question):
The method returns the answers from friend referring to question module.

3.6 Class FiftyFifty(Lifeline):

The class rewrites class Lifeline to define the lifeline 50:50.

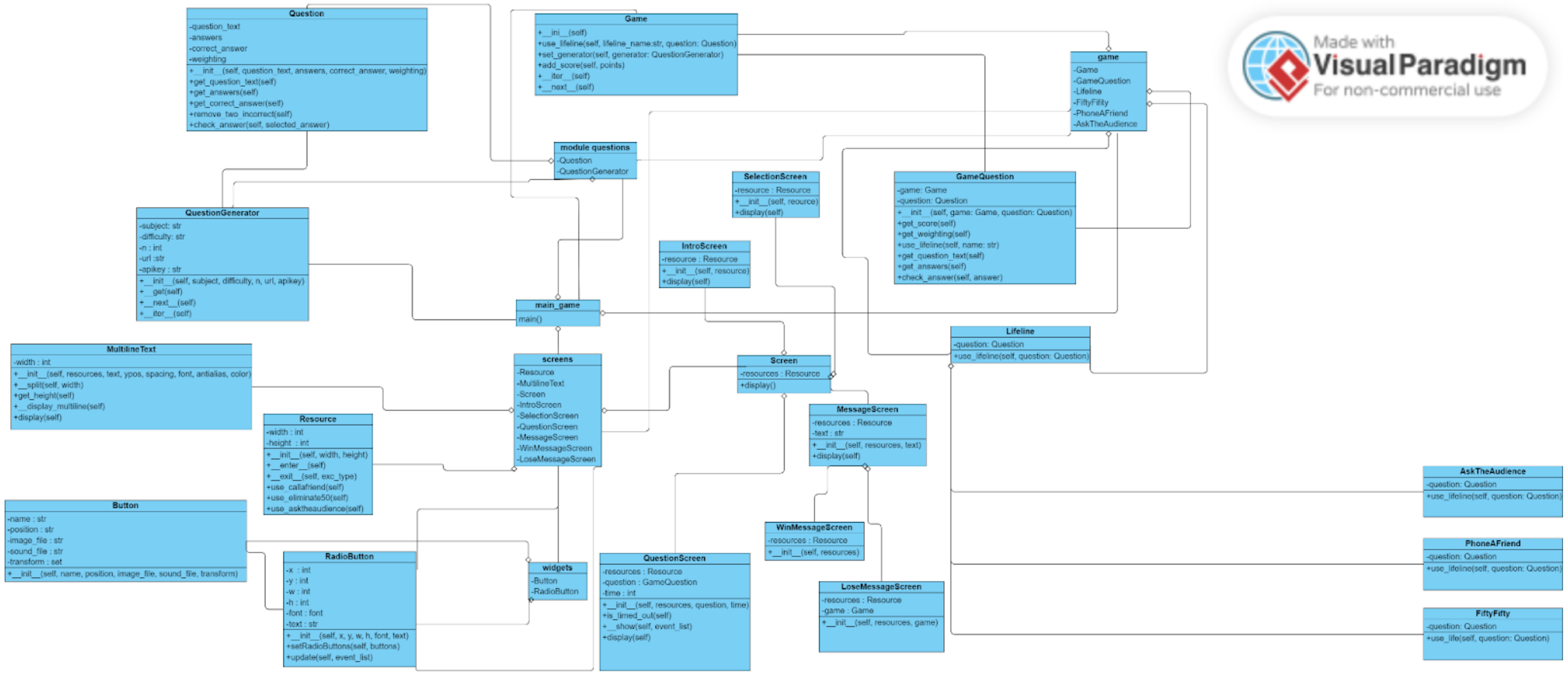
use_lifeline(self, question):
The method calls question module to remove two wrong answers referring to question module.

5. Main_Game Module

The module controls the steps of the entire program run. It imports module game, screens and questions, which explains my first line of UML diagram.

UML diagram

(Unfortunately, if the image is too blurry, attached below is the link to the visual paradigm canvas https://online.visual-paradigm.com/share.jsp?id=333232313337322d31)


Question
-question_text
-answers
-correct_answer
-weighting
+__init__(self, question_text, answers, correct_answer, weighting)
+get_question_text(self)
+get_answers(self)
+get_correct_answer(self)
+remove_two_incorrect(self)
+check_answer(self, selected_answer)
Game
+__ini__(self)
+use_lifeline(self, lifeline_name:str, question: Question)
+set_generator(self, generator: QuestionGenerator)
+add_score(self, points)
+__iter__(self)
+__next__(self)
game
-Game
-GameQuestion
-Lifeline
-FiftyFifty
-PhoneAFriend
-AskTheAudience
Made with Visual Paradigm For non-commercial use
module questions
-Question
-QuestionGenerator
SelectionScreen
-resource : Resource
+__init__(self, reource)
+display(self)
GameQuestion
-game: Game
-question: Question
+__init__(self, game: Game, question: Question)
+get_score(self)
+get_weighting(self)
+use_lifeline(self, name: str)
+get_question_text(self)
+get_answers(self)
+check_answer(self, answer)
QuestionGenerator
-subject: str
-difficulty: str
-n : int
-url :str
-apikey : str
+__init__(self, subject, difficulty, n, url, apikey)
+__get(self)
+__next__(self)
+__iter__(self)
IntroScreen
-resource : Resource
+__init__(self, resource)
+display(self)
main_game
main()
Lifeline
-question: Question
+use_lifeline(self, question: Question)
MultilineText
-width : int
+__init__(self, resources, text, ypos, spacing, font, antialias, color)
+__split(self, width)
+get_height(self)
+__display_multiline(self)
+display(self)
screens
-Resource
-MultilineText
-Screen
-IntroScreen
-SelectionScreen
-QuestionScreen
-MessageScreen
-WinMessageScreen
-LoseMessageScreen
Screen
-resources : Resource
+display()
Resource
-width : int
-height : int
+__init__(self, width, height)
+__enter__(self)
+__exit__(self, exc_type)
+use_callafriend(self)
+use_eliminate50(self)
+use_asktheaudience(self)
MessageScreen
-resources : Resource
-text : str
+__init__(self, resources, text)
+display(self)
AskTheAudience
-question: Question
+use_lifeline(self, question: Question)
Button
-name : str
-position : str
-image_file : str
-sound_file : str
-transform : set
+__init__(self, name, position, image_file, sound_file, transform)
WinMessageScreen
-resources : Resource
+__init__(self, resources)
RadioButton
-x : int
-y : int
-w : int
-h : int
-font : font
-text : str
+__init__(self, x, y, w, h, font, text)
+setRadioButtons(self, buttons)
+update(self, event_list)
widgets
-Button
-RadioButton
QuestionScreen
-resources : Resource
-question : GameQuestion
-time : int
+__init__(self, resources, question, time)
+is_timed_out(self)
+__show(self, event_list)
+display(self)
LoseMessageScreen
-resources : Resource
-game : Game
+__init__(self, resources, game)
PhoneAFriend
-question: Question
+use_lifeline(self, question: Question)
FiftyFifty
-question: Question
+use_life(self, question: Question)