Interactive Game Project

1. Presentation

This program is about playing three games sequentially. The player needs to win the current game to go to the next one. A board is displayed to show the progress of the player after each step in the game.

```
<You are here>
Random Question ~~~~~~~~~~~~~~~~~~~~~~~~~~~
Key Encryption ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Cracking the Password ~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 1. Board representing the status of the game

2. Scenario of the Game

A main menu is first displayed where the program asks the player his/her name. The player should type 1 to start the game or -99 to stop the game.

The game starts with a screen of waves and a position of the boat at the first shore.

A beginning music announces the beginning of the game.

Then the user should successfully pass the three key steps of the game. After passing a key step a winning music should be played and a version of the board should be displayed with the new position of the player. You should move the sign “You are here” into the new location. A loop should be used to implement the board. If you don’t use a loop, this part’s grade will not be awarded.

If the player is unsuccessful, the game should play a loosing music and an error message like “You lost the game …” (You can be creative here). The three key steps of the game are explained below.

2.1 Random Question

After every successful move, the new position of the player will be showed.

The first step in the game is a random question. You should prepare a battery of five questions of your choice along with their answers. The questions and their answers should be stored in a dictionary. Using the random function ***randInt***, you should ask a random question and compare the user’s answer to the answer stored in the dictionary. Asking a random question means that in every different run of the game a potentially different question will be asked.

2.2 Key Encryption

This step consists of the two steps. The game generates a sequence of **five** random characters (the encrypted sequence). The sequence of characters is presented to the user. The user should find the unencrypted version of the sequence. The game takes the answer of the user and compares it with the unencrypted version of the sequence. If the answer of the user matches the correct answer, then the user wins this step, otherwise, he/she loses. We assume the user knows the key to the encryption. You should use a dictionary for the encryption. A function to generate the random sequence should be implemented.

Sample run

Scenario 1

Gard: Hello, Here is your magic word. Could you unscramble it? AXKZH
Type your answer below.
Player: ZCPAS
Gard: Congratulations. Your answer is correct.

Scenario 2

Gard: Hello, Here is your magic word Could you unscramble it? BXKEX
Type your answer below.
Player: TYMDK
Gard: Wrong answer. See you later while being eaten by my allegators. HuhHuhHuh Houuuu

2.3 Cracking the Password

Finally, the player needs to guess the password. The game should generate the password. The password is a random 3-digit number with no repeating digits. After every clue, the game will tell you how accurate your guess was using a card system as follows:

- If it displays you a **red** card, it means that none of your three guessed digits is correct.
- If it displays an **orange** card, this means that one of the digits is in the secret number but this digit is in the wrong place.
- A green card means that one digit is in the secret number and is in the right position. Consequently, even when you get a green or orange card, you still do not know which digit in your guess is the correct one.

The gatekeeper may display multiple cards if need be. No repetition of red is needed obviously.

Sample code run

I am thinking of a 3-digit number. Try to guess what it is. Here are some clues:

<u>When I say:</u>	<u>That means:</u>
Orange	One digit is correct but in the wrong position.
Green	One digit is correct and is in the right position.
Red	No digit is correct.

I have thought up a number. You have 10 guesses to get it.

```
Guess #1:
123
Green
Guess #2:
453
Orange
Guess #3:
425
Green
Guess #4:
```

326
Red
Guess #5:
489
Red
Guess #6:
075
Green Green
Guess #7:
015
Green Orange
Guess #8:
175
You got it !!!! You won a chest full of gold and you may pass to the next game. Be prepared, you have a princess to save. Have a nice play!

3. Breakdown of the Project

Below is the breakdown of the functions to be implemented. You need to guess the parameters based on the game and function description below. You may implement any other helping functions.

main()	This is the function that calls the mainGame.
mainGame()	This function is called by the main function. It asks if the user wants to play and runs an infinite loop. Then it should call the functions of the games and display the board or the error message depending on the situation.
askRandQuest()	Should use a dictionary of question answer and should use rand int to display the random question. You may implement a sub function ***askQuestion*** that collects the user's answer and compare it to the answer in the dictionary that you may give it as a parameter to this function. The function **askRandQuest** should return 1 if the answer is correct and 0 if not.
drawboard()	Draws the board in figure 1 with updating the position of the player. Takes the number of successful steps as a parameter. This function should use a loop that prints the string <You are here>, depending on the parameter number of steps provided to the function.
playSound()	This function should play the sound file give as a parameter. You should use a winning sound file when the user gives a correct answer and a loosing sound file when the user gives an incorrect answer.
generateRandSeq()	Returns a string made of *n* characters using a loop. N is provided as a parameter to the function. This function takes a list of the chars in English and should use randint to generate the index of the random character. It should make sure no char is repeated by using an infinite loop. This function should return the sequence of characters.
KeyEncryption()	This function call ***generateRandSeq()*** to get the sequence of five random characters. Then it should ask the user to provide his/her unencrypted version of this string. Then using the encryption dictionary, where the key is the char and the value is encrypted version, you should get the encrypted version of the sequence provided by the user. After comparing

	the encrypted version generated by the game and the encrypted version provided by the user, if they match this function should return 1, and 0 if not. The encryption dictionary can use the reverse order of the chars in English.
getEncrypted()	Takes the encryption dictionary and the unencrypted version of a string and returns the encrypted version of the string.
generateEncDico()	Should take an empty dictionary and the list of characters in English. Should fill the dictionary with the chars as key and chars from the reversed list as values. For example, you should have *a* as a key and *z* as its value, and so on.
CrackPassword()	Displays the messages above and calls ***generateRandSeq()*** to generate the sequence of 3 random chars. Then uses a loop that iterates **up** to 10 times. If the user cracks the password before or at 10 iteration the loop should be stopped and the function should return 1. If no correct answer is found the game returns 0. When the user provides a new number. This number should be inspected digit by digit (it is better to represent the number as a string of characters). For example, the number 423 can be parsed into the three digits: 4, 2, and 3. Every inspected digit with its position should be sent to a function called **inspectDigit()**. This function takes the return of ***inspectDigit()*** and displays a message accordingly.
inspectDigit()	This function takes a digit (a character), its position in the number provided by the user and the secret number generated by the game. If the digit matches the char in the same position then we return the char 'g' for green. If the char exists but is in a different position, then we return 'o' for orange. Otherwise, the function should return 'r' for red. For example, if the char we are checking is 'x', its position in the user number (which is a string) is 2 and the secret password is "abc". First, we check if *x* is equal to c, which is in position 2 in the password. Since, it does not match, we check if *x* is equal to *a* or *b* (the chars in positions other than 2), since it does not we return *'r'*.

4. Reading Sound Files and Time Measurement

Below are two websites where you can download sound files. There are many others on the web.

http://www.soundjay.com/button-sounds-1.html

http://soundbible.com/tags-splash.html

- To play a sound file:

```
    import winsound
    file = "button-31.wav" # we assume the file is in the same folder as the program
winsound.PlaySound(file, winsound.SND_FILENAME)
```

5. Written Report

Every team should provide a written report (about 4-5 pages). The report should has the following parts:

1. Introduction: you describe the goal of the work and what your program should achieve.
2. Program presentation: you should present how you developed the software and what you did to overcome the challenges. Using some diagrams is not a bad idea.
3. Labor sharing: this should look like a table where you state who did what. In your first meeting you should agree on who should implement which functions. It is better to go game by game. Once the functions are implemented, you should have an integration and testing meeting where you put the functions together and see whether the code runs properly.
4. Conclusion: where you talk about what you achieved in this project and how you would improve your work in the future.

Every student should write the part of the report that describes his/her code. If a student writes the entire report and does not coding he/she will get zero.

6. Oral Presentation

The oral presentation is about showing what you have done in the project. Using slides is recommended but not required. A demonstration of a running of the program is required. You should plan a scenario of the demonstration ahead of time to show the different features of your program. All the students should be involved in the oral presentation. Every student should present his/her part. He/she should answer the questions about his/her part.