Games with PyGame

Bratucu Ana-Maria

Diaconu Andreea-Teodora

Gherasim Miruna-Alexandra

Matei Cristina

1231B

Table of Contents

1. Introduction

We live in a high technological world, everything we own is leading into having a computer inside it. As so, we are needing any kind of human computer interaction to fulfill the technological manner of our objects.

Technology is not only trying to simplify our lives, but is a very good source of entertainment. We, as humans, can spend hours, even days playing simple, sometimes stupid, games. The technology allures us into spending time using it. But a good game is not completed without a good human-computer interface. To be more concise, when something attracts us, looks good it guarantees more time using it.

This is the main reason why human-computer interfaces are so important. Technology is no longer someones' garage project, it has became an industry of thousands of dollars, and where the money lays, the stakes in everything being better increases.

Our proof of how important this subject comes in a collection of silly games developed using python and pygame module.

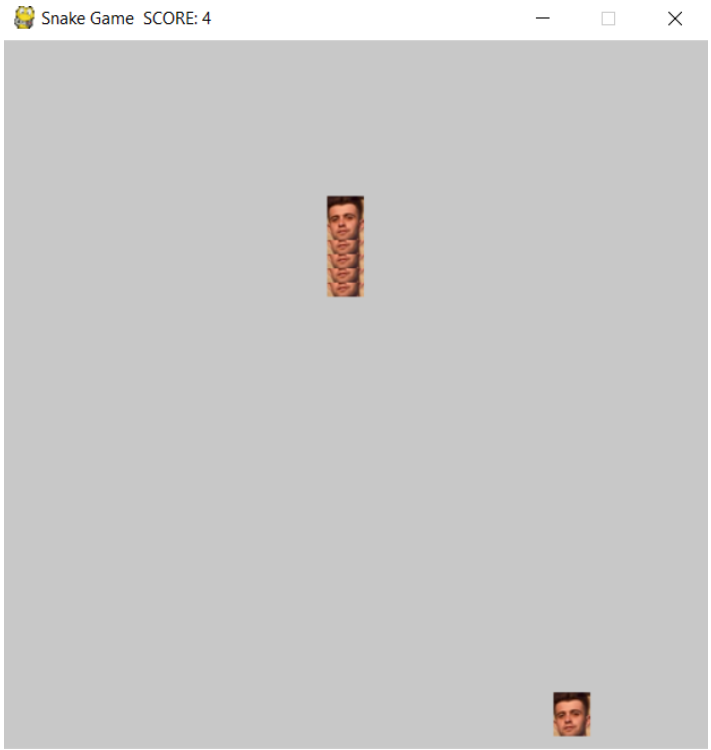
One of our games is the classic Snake, that made the world go mad and play it for hours, despite the simplicity of it.

The other one, is still a simple one, a word guesser based on input spoken by the user.

These two simple games prove how powerful the human-computer interaction actually is. You can spend hours playing these games because they bring you joy. It's no longer just a stale object standing on a desk or on our hands, it's a powerful machine that can provoke human emotions.

2. Implementation

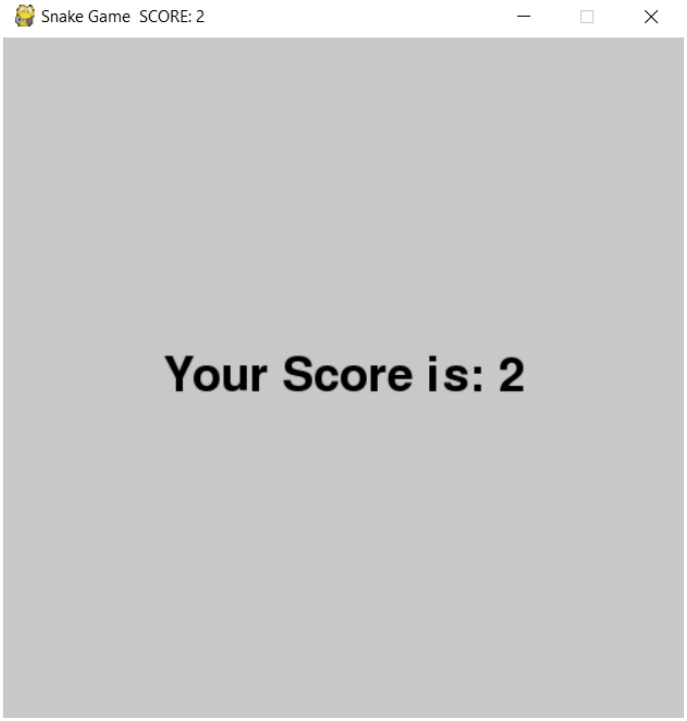
2.1. Snake – controlled by keyboards



The first game we developed was Snake, yes, the classic Snake. This game is controlled by keyboards, more specific by arrows. The score increases each time the snake hits a block (*collision_with_apple)* , and the block is added to its length. The game is lost when you hit a boundary or hit yourself, and we made two functions to implement this condition: *collision_with_self* and *collision_with_boundaries.*

Collision_with_self checks if the head of the snake's position coincides with any other position of its body and returns True or False. The function *collision_with_boundaries* verifies if his position is bigger than 500 or less than 0, because our window has 500px. In order to verify if the snake hits itself, we

take all current positions of the snake and if the head of the snake coincide with one of the current positions, we return true and the game ends. In the function *play_game*, we make the movements. Therefore, we check what was the last key pressed because you can't turn back to where you came from and we want to create continuity to the game. So, the new key pressed (that represents a position in which the snake moves) will become the last key pressed and so on. This way, the snake changes its direction with a length of 10px. If we lose (the snake touches the boundary or runs into itself), the game ends and our score appears in the window for 2 seconds before it closes.



2.2. Snake – controlled by speech

In order to identifying a speaker's intent we need to use a package SpeechRecognition.

Recognizing speech requires audio input, and SpeechRecognition makes retrieving this input really easy. Instead of having to build scripts for accessing microphones and processing audio files from scratch, SpeechRecognition will help us.

This game is very similar to the first one, except the movement part. In this case, we won't use arrows, we will use spoken words. We imported a library from Google to recognize speech named speech_recognition. In this implementation, we have an extra function called recognize_speech_from_mic which keeps what we say in the variable word["transcription"], where word is the variable that stores the result of the function recognize_speech_from_mic. Now, in the function play_game, we verify what was stored in the variable response and compare it with "left", "right", "upside" or "down". Based on this, we change the direction of our snake, by changing the value of button_direction. The snake changes its direction with a length of 10px.

2.3. Guessing Game - controlled by speech

This game is a small game that picks a random word from a list and gives the user three attempts to guess the word.

```
I'm thinking of one of these words:
apple, banana, grape, orange, mango, lemon
You have 3 tries to guess which one.

Guess 1. Speak!
You said: banana
Incorrect. Try again.

Guess 2. Speak!
You said: grape
Incorrect. Try again.

Guess 3. Speak!
You said: mango
Sorry, you lose!
I was thinking of 'lemon'.

In [4]:
```

The recognize_speech_from_mic() function takes a Recognizer and Microphone instance as arguments and returns a dictionary with three keys. The first key, "success", is a boolean that indicates whether or not the API request was successful. The second key, "error", is either None or an error message indicating that the API is unavailable or the speech was unintelligible. Finally, the "transcription" key contains the transcription of the audio recorded by the microphone.

The function first checks that the recognizer and microphone arguments are of the correct type, and raises a TypeError if either is invalid.The listen() method is then used to record microphone input.The adjust_for_ambient_noise() method is used to calibrate the recognizer for changing noise conditions each time the recognize_speech_from_mic() function is called.Then recognize_google() is called to transcribe any speech in the recording.A try...except block is used to catch the RequestError and UnknownValueError exceptions and handle them accordingly. The success of the API request, any error messages, and the transcribed speech are stored in the success, error and transcription keys of the response dictionary, which is returned by the recognize_speech_from_mic() function.

We have a list of words, a maximum number of allowed guesses and a prompt limit are declared:

```
WORDS = ['apple', 'banana', 'grape', 'orange', 'mango', 'lemon']
```

Next, a Recognizer and Microphone instance is created and a random word is chosen from WORDS:

```
recognizer = sr.Recognizer()
microphone = sr.Microphone()
word = random.choice(WORDS)
```

A for loop is used to manage each user attempt at guessing the chosen word. The first thing inside the for loop is another for loop that prompts the user at most PROMPT_LIMIT times for a guess, attempting to recognize the input each time with the recognize_speech_from_mic() function and storing the dictionary returned to the local variable guess.

If the "transcription" key of guess is not None, then the user's speech was transcribed and the inner loop is terminated with break. If the speech was not transcribed and the "success" key is set to False, then an API error occurred and the loop is again terminated with break. Otherwise, the API request was successful but the speech was unrecognizable. The user is warned and the for loop repeats, giving the user another chance at the current attempt.When the inner for loop terminates, the guess dictionary is checked for errors. If any occurred, the error message is displayed and the outer for loop is terminated with break, which will end the program execution.

If there weren't any errors, the transcription is compared to the randomly selected word. The lower() method for string objects is used to ensure better matching of the guess to the chosen word. The API may return speech matched to the word "apple" as "Apple" or "apple," and either response should count as a correct answer.

If the guess was correct, the user wins and the game is terminated. If the user was incorrect and has any remaining attempts, the outer for loop repeats and a new guess is retrieved. Otherwise, the user loses the game.

3. Conclusions

Human-computer interaction is a very interesting subject that is spreading worldwide faster than any other field.

In our project, we demonstrated that we can create human-to-machine interaction even with the simplest of things. Not only that it is possible to control computer objects with the help of four buttons and even you own voice, but also it is possible to play games against the computer by guessing what the computer is thinking about.

We wanted to add a touch of humour to our snake game, so we added a picture of our colleague instead of the basic snake and apple.

However silly or simple our games may seem, their implementations were fairly complex and by using Google's package of SpeechRecognition, we made it more fun and dynamic. To further improve our games, we think it would be useful a more friendly interface and a better design. Also a more important aspect would be to create a continuous speech recognition function, so that the snake game will have a more fluid continuity to its movements, instead of waiting for our vocal input before making the next move.