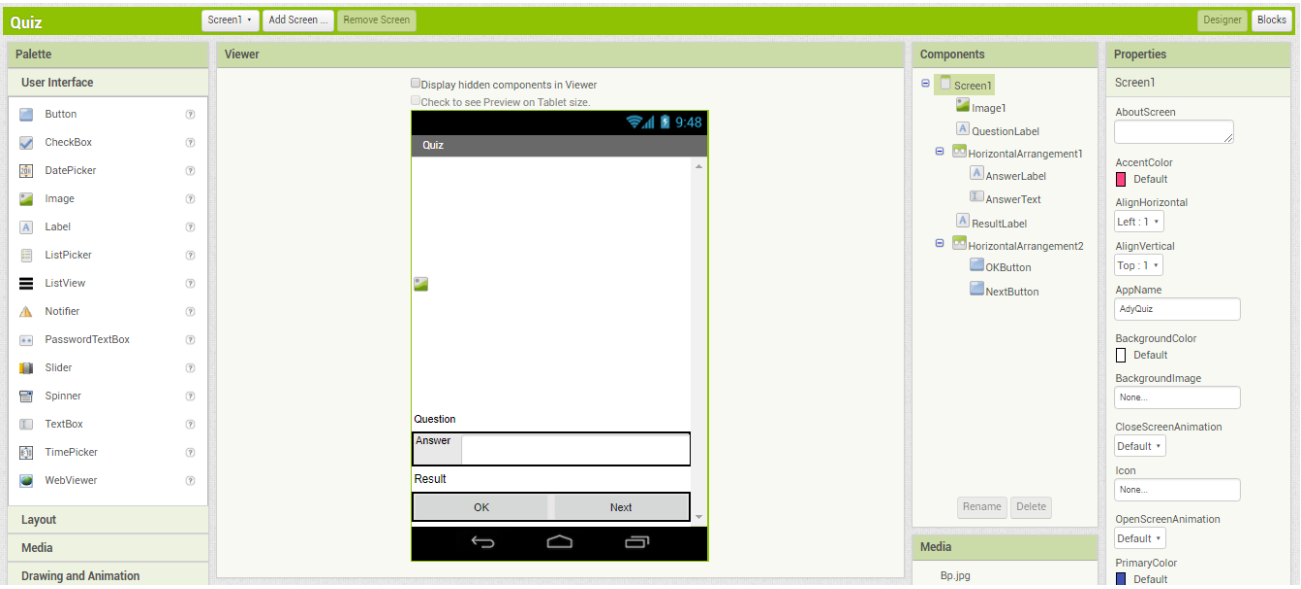
Task 1 | **GIMP** | ***"Filters"***

- By using filters, we can set decorative effects to our image to make it looks better.
  - If the image consists of one layer only, and there is no selected area, the filter will modify the whole image.
  - If the image consists of more layers, and there is no selected area, the filter will modify the currently active layer only.
  - If there is an area selected on the image, the filter will modify the selected area only.
- Choose an image on the following website: https://apod.nasa.gov/apod/archivepix.html
- Download the chosen image to your computer and create your own, totally unique image by adding one or more filters to it. Click on the *Filters* option and have fun. 😊
- You can find some examples for using filters below.

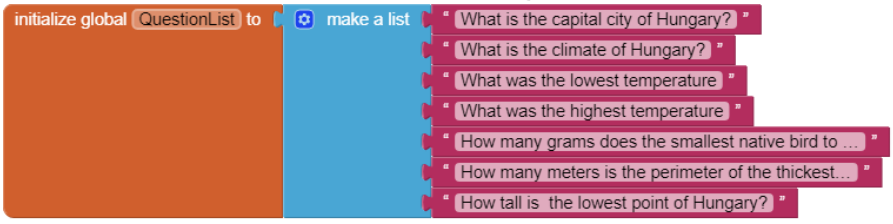
- Save your project as *Filter01*, *Filter02* etc. in *.xcf* and also export the image in *.png* format.
- Find out more about *Filters* by visiting: https://docs.gimp.org/en/filters.html

Task 2 | **MIT App Inventor 2** | ***"Quiz 1"***

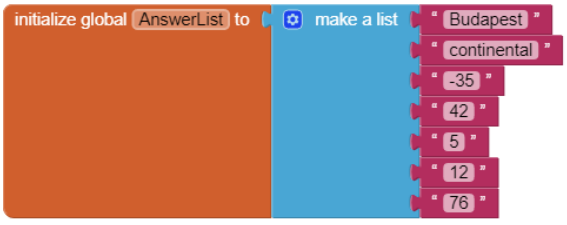
- Here are the components for the *Quiz 1* app, as shown in the *Component Designer:*



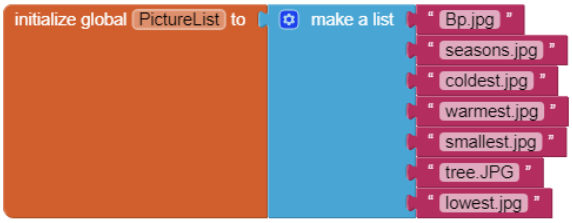
- Create 3 variables consisting of 3 lists.
  - In the first one, have to be stored the *Questions*.



  - In the second one, have to be stored the *Correct answers*.



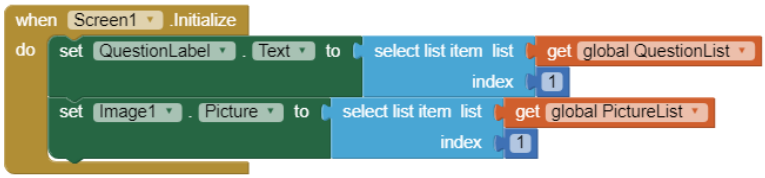
  - In the third one, have to be stored the *Images* for the questions.



- Create 2 more variables.
  - One of them has to store the number of the current question.

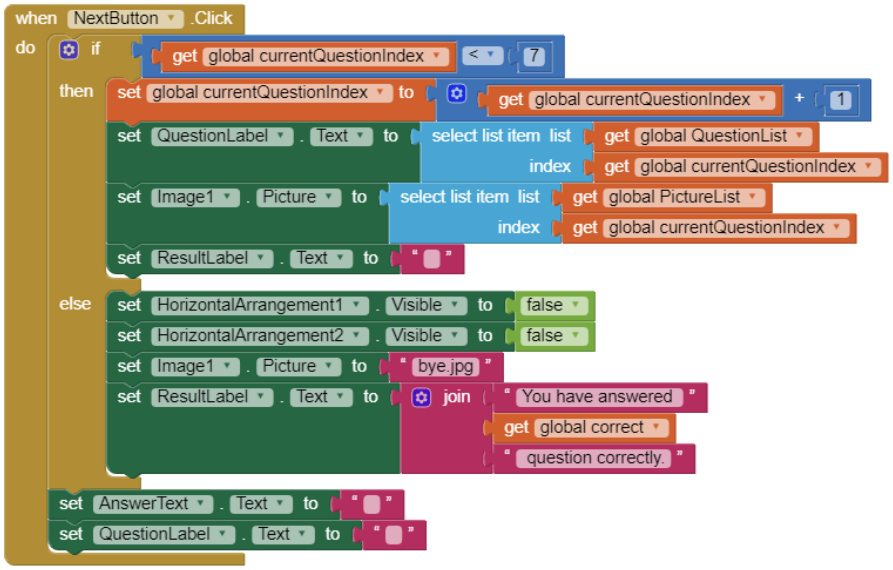
  - The other one has to store the number of the correct answers.

- At the start of the application the *QuestionLabel* has to display the first element of the *QuestionList* and also the belonging image.



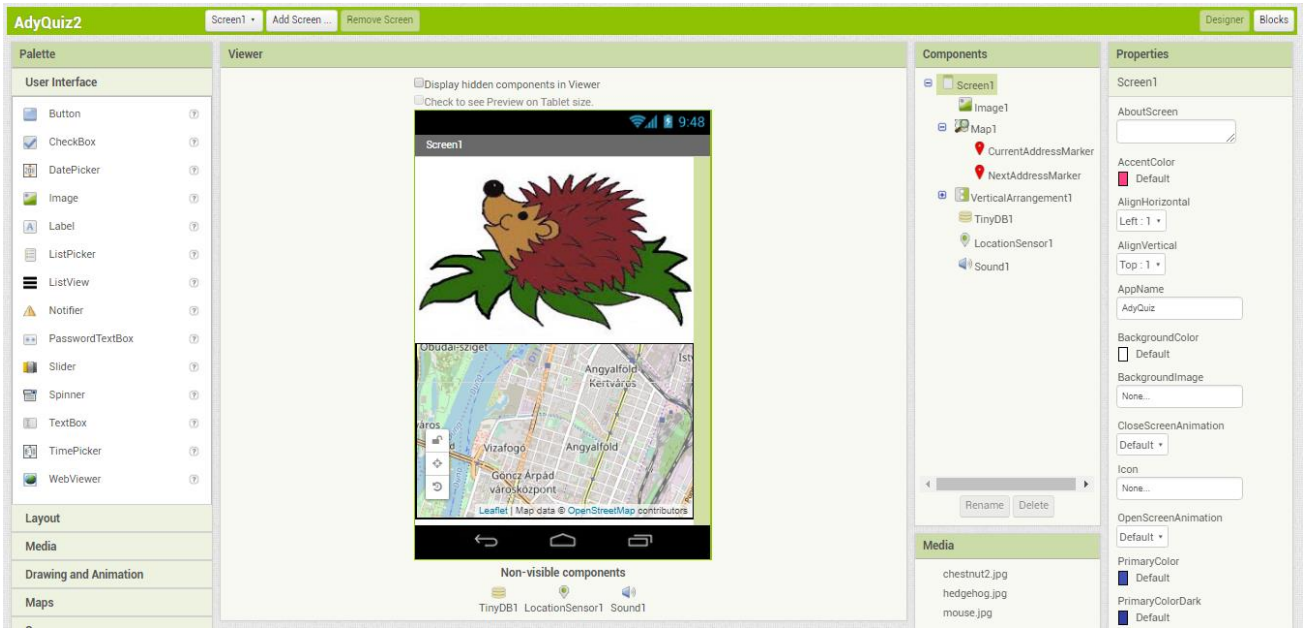
- When the *OKButton* is clicked, it has to analyze if the entered text is in accordance with the current element of the *AnswerList*.
    - If yes, the *ResultLabel* has to show the correct answer and increase the number of the correct answers by adding plus one point.
    - Else, it has to show the *"Think about it!"* text.

- When the *NextButton* is clicked, it has to analyze if the number of the question is smaller than the index of the last element of the *QuestionList*.
    - If yes, it has to increase the number of the questions by adding one more, so the *QuestionLabel* has to show the next element from the *QuestionList* and also the belonging picture.
    - Else, it has to turn off the *AnswerTextBox* by setting the *HorizontalArrangement1* and *HorizontalArrangement2* (which contain the buttons) to invisible.
        - It has to show the *"BYE"* image and the number of the correct answers.
        - And it has to set the *AnswerText* and the *QuestionLabel* to empty.



Task 3 | **MIT App Inventor 2** | ***"Ady Quiz"***

- Here are the components for the *Ady Quiz* app, as shown in the *Component Designer:*



- Declare the following variables:
  - *currentQuestionIndex:* shows where we currently are.

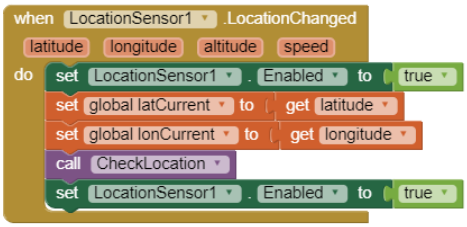
  - *currentQuestionText:* elements of the database (by tags) which are set in list.

  - *questionData:* the data of the current point will be stored here.

  - *lat* and *lon:* the coordinates of the destination.

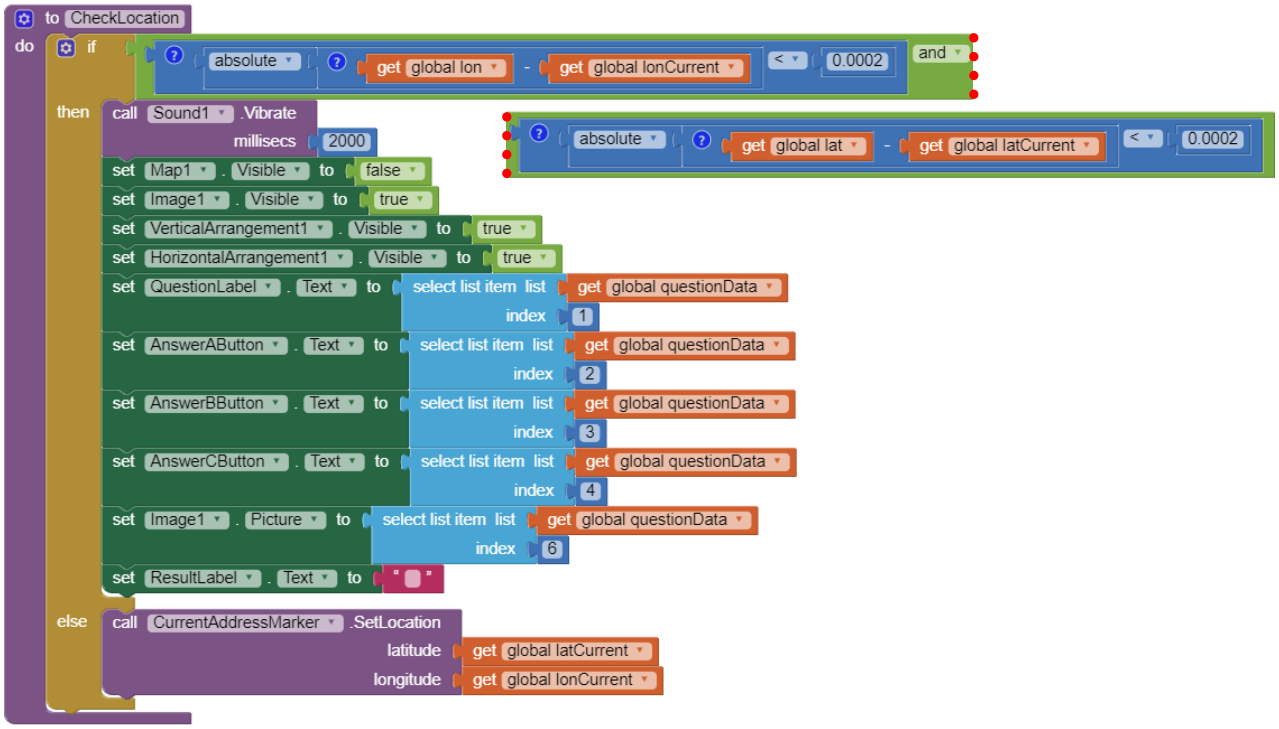
  - *latCurrent* and *lonCurrent:* the coordinates of the player's current positions.

- Turn the *LocationSensor* on and define the current position of the player.

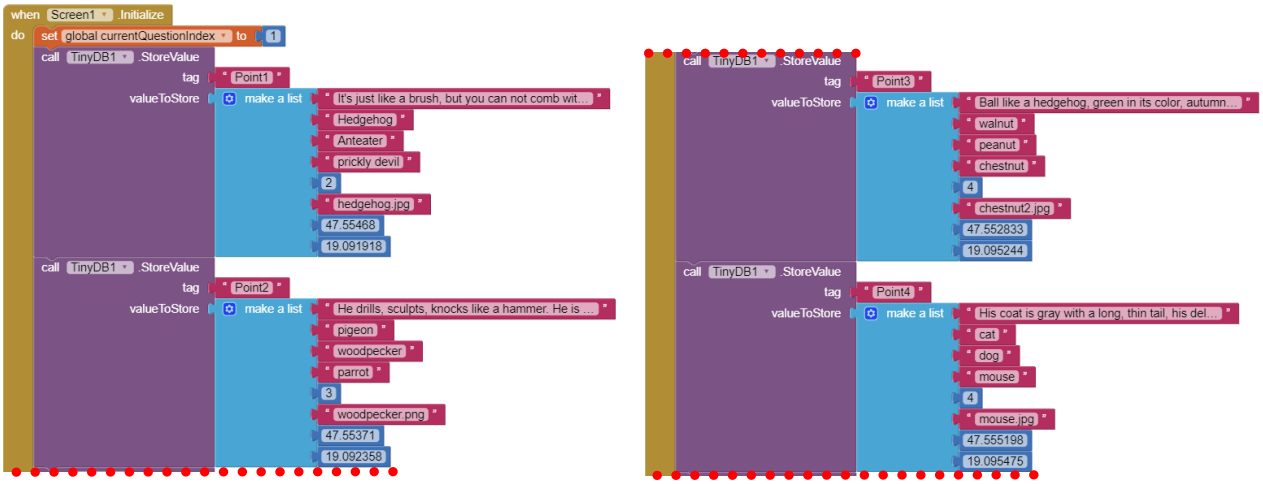


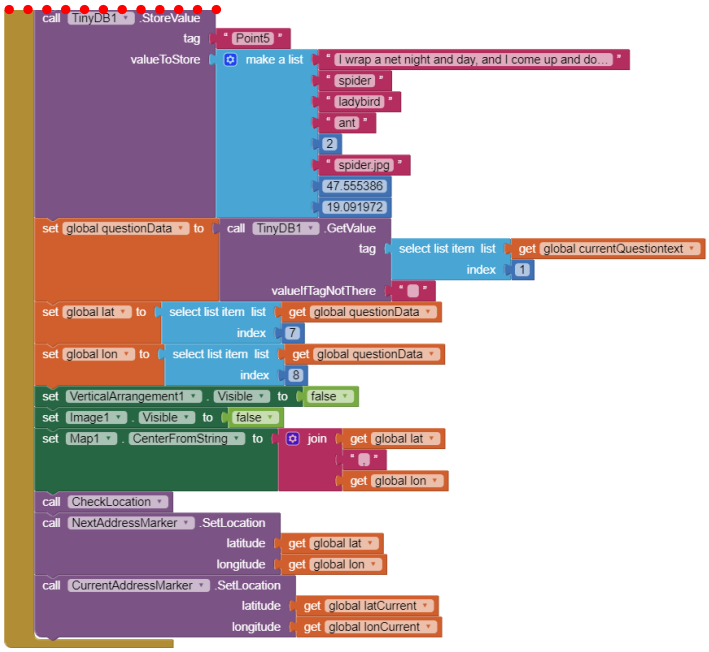
- Check the position of the player to know how far the player is from the destination.
  - If the player is close to the right place the phone has to be fluttering, the map has to disappear and the picture, belonging to that point, has to show up.
  - Then the *QuestionLabel* and the multiple choice answer buttons have to show up.
  - The *ResultLabel* has to be set blank.

- If the player isn't close to the point, the map has to show the current position of the player.

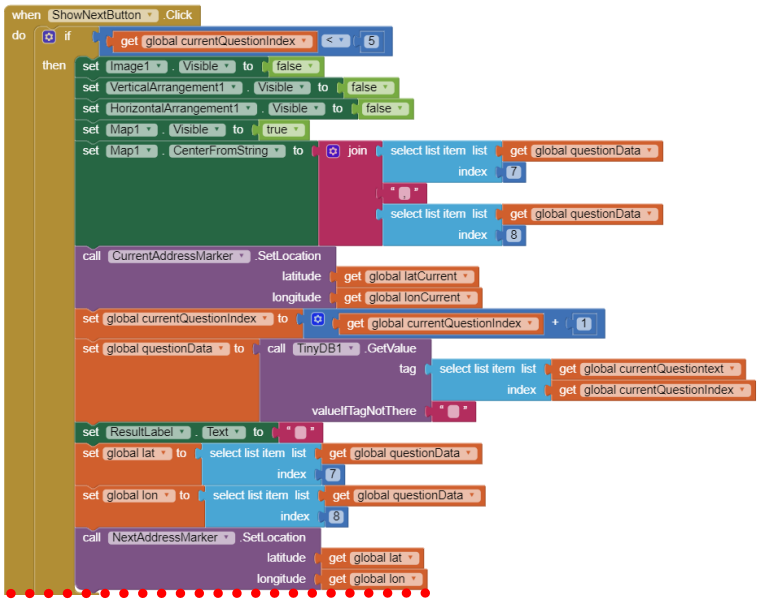


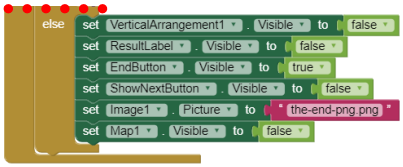
- At the start of the application, the value of *currentQuestion* has to be set to *1*.
  - Store the items of the *Points* with the *StoreValue* procedure.
  - The first one has to be the defining tag, then create a list consisting of the belonging question, the optional answers, the index of the correct answer, the image, and the GPS coordinates.
  - Store the items to each *Points*.
  - Set the value of the *questionData* variable to the data of the current point.
  - Set also the coordinates of the destination.
  - Set the image and the buttons to invisible and set the map to visible. Set the center of the map to be the destination exactly.
  - Call the *CheckLocation* procedure.
  - Create markers on the map to show the destination and the current position.





- If the *ShowNextButton* is clicked, the current position of the player has to be checked.
  - If the player hasn't approached the last point the visibility of the image, the question and answers have to be set to *false*.
  - Set the map to visible and the center of it to be the current point exactly. Create a marker on the map to show the current point.
  - Then, switch to the next question. The data of it has to be retrieved from the database and the value of the *questionData* variable has to be rewritten.
  - The text of the *ResultLabel* has to be set to blank.
  - Retrieve and store the coordinates of the next destination and a marker has to show its exact location on the map.
    - If the player has approached the last point, the visibility of the image, the question and the answers have to be set to *false*.
    - Then, set the *EndButton* to visible and show the image with the name *'the-end-png.png'*.





- Check the answer if it is correct or not with the same method at the 3 buttons.
  - If the button clicked is in accordance with the correct answer stored, the *ResultLabel* has to show *'Correct!'*.
  - Otherwise, it has to show *'Check it up!'*.