Apiza: The API Assistant Study

Description

In this study, you will be asked to implement a number of features in a simple game program using the Allegro API. To assist you with these tasks, you will be connected with our experimental virtual assistant, **Apiza**. Apiza is an AI monitored by a human researcher that can carry out organic, natural-language conversation. Apiza will learn as the conversation progresses and fit its response to the context of the conversation.

As you work through these tasks, you will have to make use of functions and constants from the Allegro API. When you have any questions about the API, such as determining an appropriate function or learning about function parameters, we ask that you direct your question to Apiza. Apiza is capable of discussing both high-level and low-level functionality.

If you have other general programming questions, feel free to consult internet resources (Google, StackOverflow, etc). However, avoid mentioning Allegro or any of its specific components in your queries, as all of those questions should be directed towards Apiza.

If you run into any issues, please inform the study supervisor. You will be compensated for your participation, so take your time and do your best. It is alright if you do not complete all of the tasks in the allotted time.

Set-up

Before beginning, you should have the provided VirtualBox appliance imported in VirtualBox. Start up the virtual machine and login to the "Participant" account using the password "password". Login to the Apiza workspace in Slack on either the Virtual Machine or the host machine (whichever is more convenient). Once the study coordinator says to begin, click on the user "Apiza" in slack to begin chatting with the virtual assistant.

The project you will be working on is found in "`~/Documents/Tasks/`". The only file you will need to modify is "`~/Documents/Tasks/src/allegro_tasks.c`". All necessary headers are already included. Using whichever text editor you prefer, complete the tasks described on the next pages.

Tasks

To test the program, navigate to `~/Documents/Tasks/` in the terminal and run `./build_and_run.sh`. You will see that some basic features have already been implemented. As you work through each task, test the program to confirm that your changes work as expected. The tasks may be completed in any order, unless otherwise specified.

1. Add keyboard support for the player character.
   - When the player presses an arrow key on the keyboard, the player character should move in the corresponding direction.
   - When the player releases an arrow key, the player character should stop moving in that direction.
   - **Hint:** The `movePlayer` function is called every frame in the game loop, taking the `player` character and the boolean array `keys` as parameters. Update the `keys` array appropriately when an arrow key is pressed or released to have the character respond to the keys.

2. Add a "game over" sound effect when the two characters collide.
   - Whenever the player character collides with the enemy, the sound effect found in the `AUDIO_SAMPLE_PATH` should be played once.
   - Note: the collision detection is already in place in the game loop

3. Display the score in upper-left corner and the highscore in the upper-right.
   - The scores should be written in the font found in the `FONT_PATH`.
   - The score should be left-aligned in the upper-left corner and the highscore should be right aligned (neither should go out of frame when the values get large).
   - Both visible scores should be redrawn every frame.
   - Note: The functionality to keep track of the `score` and `highscore` variables is already implemented

4. Replace the default character bitmaps with .png images
   - The player and enemy character bitmaps should be replaced with the images found in the `BITMAP_PLAYER_PATH` and `BITMAP_ENEMY_PATH`, respectively.
   - The images should be scaled to the size of their respective bitmaps (the fly.png sprite should be drawn at 32x32 px, and spider.png should be 96x96 px).
   - **Hint:** Prepare the bitmaps before entering the the game loop by drawing the scaled images onto the existing `player_bitmap` and `enemy_bitmap`. Then when the frame is updated, the `player_bitmap` and `enemy_bitmap` can be drawn as normal.

5. Rotate the player bitmap based on the direction of movement.
   Note: Task 4 should be previously completed

    - While the player character is moving left, right, up, or down, the player bitmap should be rotated accordingly, such that the top of the bitmap is pointing in the direction of movement.
    - When the character stops moving in a direction, it may either stay facing that direction or revert to the default orientation.
    - If the character is moving both horizontally and vertically, it may either be rotated diagonally or toward just one of the directions.

6. Pause the game when the player clicks in the window.
    - When the player clicks the mouse in the game window while the game is running, it should pause (no movement or speed increases). When the player clicks while the game is paused, it should resume
    - Note: the game can be "paused" and "resumed" by stopping and starting the **`timer`** and **`speed_timer`** variables.

7. Make the window resizable.
    - Allow the game to be resized while by dragging the corners of the window.
    - You may either update the positions and/or sizes of the characters relative to the window, or let them retain their absolute values.
        - Whichever you choose, the characters' movement must still be bounded by the edge of the window frame.