

CS1112 Fall 2010 Project 6 Part 1 due Thursday 12/2 at 11pm

(Part 2 will appear in a separate document. Both parts have the same submission deadline.)

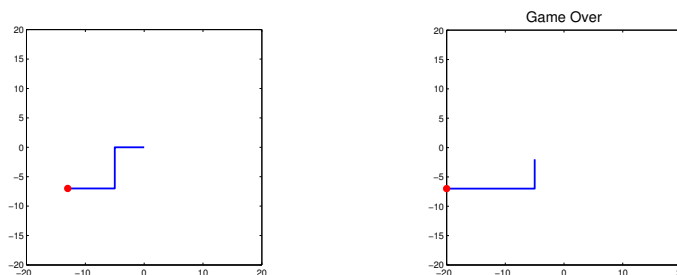
You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the project you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group.

Objectives


Completing this project will solidify your understanding of acoustic data manipulation (Part 1) and recursion (Part 2). You will also practice problem decomposition and testing as well as flex your creative muscles!

1 Go Worm

Implement a game with a “steerable” worm! The worm starts moving from the middle of the game grid and unless a player steers it by clicking on the plot, it will crash into a wall and die. The sound effects of the game include a proximity alert that sounds when the head of the worm approaches a wall and a gameover sound when the worm crashes into a wall.



1.1 General Specifications

- The game is played in a 40-by-40 grid, centered on the origin, in a MATLAB figure window. The game begins with the function call `wormGame(t)` where `t` is a positive number in the range of .01 to 1.
- The worm has 20 connected unit-length segments that are each oriented vertically or horizontally. For example, the coordinates (1,1), (2,1), (2,2), (3,2), (3,1), (4,1) can specify the shape (position) of a 5-segment worm: 
- Each segment of the worm can move only horizontally or vertically and the user steers the worm by clicking somewhere within the plot. If the horizontal distance from the worm's head to the clicked point is greater than the vertical distance, the worm should move horizontally (left or right, toward the point). Otherwise, it should move vertically (up or down, toward the point).
- Each segment of the worm moves 1 unit per `t` seconds, i.e., `pause` code execution for `t` seconds between updates to the worm's position. `t` is the sole parameter to the game function `wormGame`.
- The game ends when the head of the worm touches a wall, i.e., the edge of the 40-by-40 grid. A sound effect plays when the game ends.
- A proximity alert sounds when the head of the worm is less than 10 units from a wall. The alert has maximum volume when the head is one unit or less from a wall and zero volume at 10 units from a wall; the volume between 10 units and one unit from the wall is (you guessed it) linearly interpolated. The alert sound is *stereo*: higher volume on the left speaker when the head of the worm is in the left half of the plot and higher volume on the right speaker when the head of the worm is in the right half of the plot. (Details are given below.) Each alert sound plays up to `t` seconds.

1.2 MATLAB Code, Program Development, and Details

Download the files `wormGame.m` and `makeClickableFig.m` and read the comments and given code. You will complete the function file `wormGame` to implement this game. Function `makeClickableFig` is completely implemented to set up a figure window that responds to mouse clicks.¹

1.2.1 Program development

Don't try to do everything at once! Decompose the problem into different parts, add the parts to `wormGame` one at a time, and test that the program runs after adding each part. In past projects we decomposed the problem for you and suggested the order in which to work; this time around, in your final project, you will do much of the problem decomposition yourself. (We still provide some help, as shown below.) Use subfunctions to encapsulate independent tasks. One of the first subproblem that you need to solve in this game is how to represent the worm's position and direction in your code. *Taking the time to plan first and to test systematically after adding each detail will end up saving you time.* And it will give you a good (and fun) finished product.

1.2.2 Code layout

Some of the needed initializations are already given in `wormGame`. Other initializations necessary include the worm's initial position and direction, which are up to you. The worm's head should not start too near to a wall though! The game proceeds using iteration—at each iteration

- Detect whether there's a mouse click and therefore any change in direction
- Calculate the worm's new position and redraw the worm in the plot. When the worm's head moves in some direction one unit the body must follow. Don't let the worm get longer (or shorter) by accident.
- Play the proximity alert sound if the worm's head is near a wall as specified above.

The game ends when the worm's head touches a wall and a sound effect signaling the end of the game is played.

1.2.3 Graphics

Use `plot` to draw the worm as a line (line segments) in the figure window. Use a marker to indicate the worm's head. Matlab's default behavior is to set the axes limits to display the data in `plot` in the middle of the figure. We want the axes to have fixed limits so that the worm is seen to be moving inside a grid, instead of the grid moving around the worm “fixed” to the center. Therefore use this command to set the axes after a call to `plot`:

```
axis([-w/2 w/2 -w/2 w/2], 'square')
```

where `w` is the width and height (the grid is centered on the origin).

1.2.4 Detecting mouse clicks

Both `wormGame` and `makeClickableFig` contain the code

```
global click
```

This is a declaration to set the property of the variable named `click` to be “global,” which means that this variable is accessible from any function or script file that contains this declaration. Once `makeClickableFig` is called to set up a figure window, every time that a user clicks in that window the variable `click` stores the coordinates of the clicked point. `click` is initialized as the empty vector, so to determine whether a click has been made you can just check whether it is empty. The built-in function `isempty` can be used: `isempty(click)` returns true (1) if `click` is the empty vector. Be sure to reset `click` to empty after you've used the values of the user clicked point.

¹This is different from using the function `ginput` to collect the coordinates of clicks in a window. If a program uses `ginput`, it *waits* for the user click(s) and only after the user has completed the click(s) will the program continue executing subsequent commands. In this game the `wormGame` function continues execution whether or not the user makes a click, i.e. `wormGame` does not *ask* for clicks; it responds to clicks *if* they are made.

1.2.5 Game sounds

Implement the subfunctions `getAlertSoundData` and `getGameOverSoundData`. It is up to you what sounds to use but please exhibit judgment and good taste. You may use a `wav` file or synthesize your own sound. A file `quack.wav`, which contains the sound of a duck quack, is available on the course website for your use. For these two subfunctions you may change the input and output parameters based on your design. Be sure to revise the function comments appropriately if you change the function header. You will submit your `wav` files if you choose to use them in the game.

No matter what length the sound data returned from `getAlertSoundData` is, the actual alert sound that is played at each iteration is no more than `t` seconds. For `t` less than the play duration of the available sound data, make sure that your code selects `t` seconds' worth of data *that is audible*. Many `wav` files start and end with some period of silence, so a safe route is to select the *middle t seconds* of the sound data to play.

A sound is *mono* if its data is a 1-d array, i.e., only one column (also known as a *channel*). A *stereo* sound is represented by an M -by-2 matrix, where M is the length of the sound vector. In other words, instead of having only one vector for both speakers, we now have a vector for each speaker. The first column of the sound matrix is for the left speaker, and the second is for the right. The command `sound(M,rate)` where M is a matrix with two columns plays a stereo sound.

You must use stereo sound for the proximity alert. Our stereo sound will be based on the horizontal position of the worm's head. Define the fraction from the left edge as

$$posfrac = \frac{x - L}{R - L},$$

where x is the x-coordinate of the worm's head, and L and R are the x-coordinates of the left and right edge of the plot, respectively. The sound that should be output to the left speaker is given by

$$leftSpeakerSound = (1 - posfrac) \cdot B,$$

and the sound that should be output to the right speaker is given by

$$rightSpeakerSound = posfrac \cdot B,$$

where B is the 1-d sound vector.

1.2.6 Bells and whistles

Show us your creativity and your programming skills! Impress us with interesting sound effects, graphics, even twists in the plot (game plot, not the figure!). We will be impressed with your demonstrating your programming techniques—not calling built-in functions. So show us *your* skills and ideas. Five percent of this project's score will be devoted to the “bells and whistles” that you demonstrate. Don't ask us how much you need to do! When you write an essay you might get an 'A' if it's excellent, a 'B' if it's good, and so forth. It's the same with this small, subjective part of the score—you get more parts of the .5 points if we consider your demonstration to be impressive.

But we do give you some guidelines here . . . Download a `wav` file and play it straight? Not impressive. Take data from a `wav` file and produce some sound effect with your code? Nice! Format the figure window with your code (e.g., things you learn from Appendix A of *Insight*)? Cool! Add visual/acoustic effects with your code to lead in to the game? Wow! Add a twist to the game and demonstrate good programming technique? How fun! Contradict the specifications or implement a totally different game? Don't do it as you will get zero points for Project 6 Part 1! If you add any twist to the game, *you must clearly state the differences from our original game in a comment block in wormGame*. We hope you will have fun with this project by thinking about cool things to do, but don't fixate on the 0.5 points! Have fun!

Put your file `wormGame.m` as well as any `wav` files that your game needs in a `zip` file called `p6part1.zip` and submit it on CMS. Ask a lab consultant if you need help with zipping files.