ME 5493 Robotics @ UTSA

Mini-project #1

Single player pong game
Stage 1: due on the week of March 18 - 22;
Stage 2: due on the week of April 1 - 5

1 Goal

The overall goal is to create a single player pong game. The project has two stages (see Sec. 5 Grading before you start the project):

Stage 1: You will complete the provided code get a basic version of the pong to work. This involves programming the ball to bounce of the walls and paddle. This is needed for the second stage.

Stage 2: You will add more features into the game to make an advanced version of the pong game. This is mostly an open-ended exercise where you are required to add at least three innovations that make the game more interesting to play.

2 Understanding the game provided to you

Please download the game *basic_pong.m* from blackboard. Open the file in MATLAB and run the game without pressing any key on the keyboard. This will produce an animation that looks like the one shown in Fig. 1. You will notice that the red ball penetrates the paddle and the game stops when the red ball reaches the bottom left corner of the screen. Now run the code again but use the left/right arrow keys to move the paddle. Again, you will see that the ball is not detected by the paddle and the game ends.

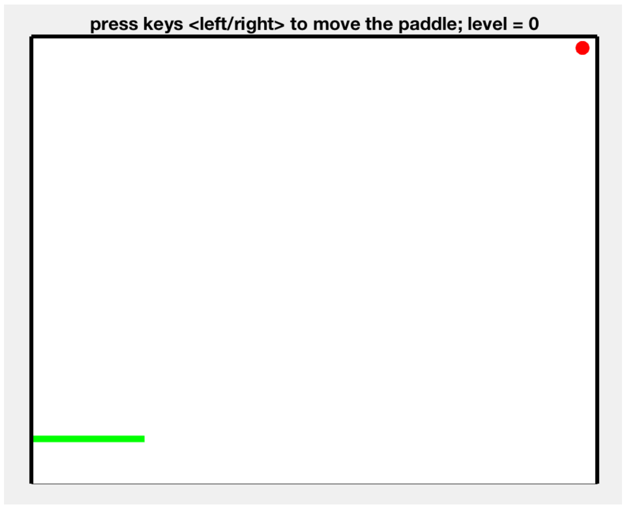
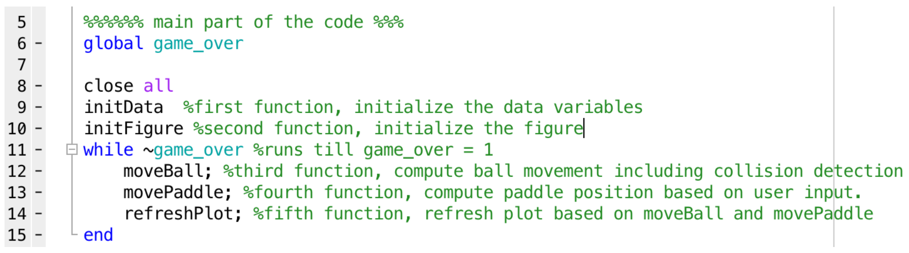


Figure 1: After running *basic_pong.m* in MATLAB you will see a figure that looks like this.

3 Stage 1: Programming the basic pong game

The main part of the code is shown in Fig. 2. It consists of 5 functions from lines 9 to 14. Your task is to program the functions *moveBall* and *movePaddle*, rest of the functions have been already programmed. More details are in the MATLAB script file inside the functions.



```
 5       %%%%%% main part of the code %%%
 6 -     global game_over
 7
 8 -     close all
 9 -     initData  %first function, initialize the data variables
10 -     initFigure %second function, initialize the figure
11 -     while ~game_over %runs till game_over = 1
12 -         moveBall; %third function, compute ball movement including collision detection
13 -         movePaddle; %fourth function, compute paddle position based on user input.
14 -         refreshPlot; %fifth function, refresh plot based on moveBall and movePaddle
15 -     end
```

Figure 2: The basic structure of the *basic_pong.m*. In order to complete stage 1, you only have to program the functions *moveBall* and *movePaddle* using the instructions inside those functions (also see Fig. 3)

Hints to help you get started: There are a series of constants that are defined using capitalized letters (e.g., WALL_X_MIN, PADDLE_WIDTH, DT). These should be self-explanatory from their names. If not, you will have to understand their meaning by skimming through the code and/or varying the constants and running the code. These constant variables as well as some other variables that are not constant (e.g., game_over, paddle_x_left, ball_x) but are needed in multiple functions are defined as "global" variables (search MATLAB global if you want to know more). These global variables are available for use anywhere in the code, so we do not have to pass them as arguments inside functions. Generally using global variables is a bad practise but here it actually quite useful as it makes the coding efficient.

- Figure 3 gives some hints on the pong board including the origin, axis, boundaries for the walls, and paddle location.
- In function *moveBall* you will have to detect if the ball has hit the obstacle, the left wall or the right wall or the top wall or the paddle. After detection, you need to impose conditions on the velocity of the ball after it hits the particular wall/paddle. More specifically, The velocity of the ball before it hits the wall/paddle is known. Let us call this: $\vec{v}^- = v_x^- \hat{\imath} + v_y^- \hat{\jmath}$ (known). The velocity after the ball leaves the wall/paddle is unknown. Let us call this: $\vec{v}^+ = v_x^+ \hat{\imath} + v_y^+ \hat{\jmath}$ (unknown). Using physics, the unknown velocity may be found from the known velocity as follows.

  1. If the ball hits the left or right wall, the velocities after collision are: $v_y^+ = v_y^-$ and $v_x^+ = -e_x v_x^-$.
  2. If the ball hits the top wall or the paddle, the velocities after collision are: $v_x^+ = v_x^-$ and $v_y^+ = -e_y v_y^-$.

  where e_x and e_y are numbers that you can choose to make the game interesting. Initially, it is suggested that you put $e_x = e_y = 1$ to get the game to work. Other suggestions are: you could keep $e_x = 1$ and change $e_y = 1 + 0.1 \times rand$ (where rand is a random number generator function) so that the ball accelerates if it hits the paddle or top wall, you could also reverse the logic to $e_y = 1$ and $e_x = 1 + 0.1 \times rand$. Feel free to choose other variants.

- In function *movePaddle* you need to ensure that the paddle stops when it touches the left or right wall.

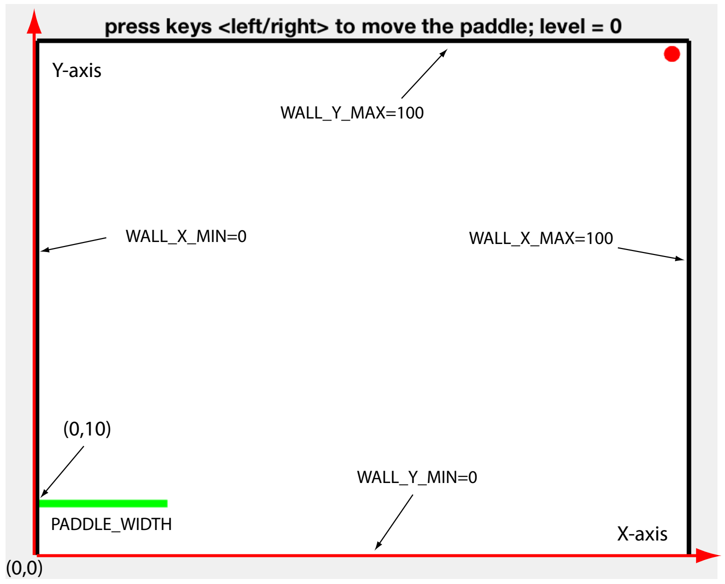


Figure 3: The pong figure with some key variables in the code.

4 Stage 2: Programming the advanced pong game

The advanced version of the game may be programmed once you are done with the basic version above. You will add *at least* three innovations to the game. One of them has to be adding a new visual element that interacts with the paddle/ball in some way (see point 1 below) and other two totally up to you. Here are some ideas but you are free to come up with your own.

1. It is mandatory to add at least *ONE* visual element that interacts with the ball or the paddle. For example, (1) you could add random stationary balls on the board, which garner extra points when hit; (2) you could add a stationary/moving horizontal bar on the board that would garner extra point if the ball hits the board; (3) you could create a two-ball pong game but note that it is difficult to play such a game unless you clearly think on how to play such a game to make it interesting (perhaps one slows down one the other is hit).

2. You could add have a scoring criteria and show it explicitly as the person plays the game. You could keep a track of the highest score and display it. For the latter, you will have to write the score to a file on your drive (search for the command “save” and “load”).

3. You could remove detection of the left and right wall so that the ball moves continuously from left to right and vice versa.

4. You could add up/down movement to the paddle in addition to the left/right movement.

NOTE: I do not recommend making the innovations too complex (e.g., creating a two-player pong) so that you are able to finish the project on time. Also, please put your name on the x-axis using xlabel("name"). Don't forget to increase the initial speed of the ball else it will be too slow to be interesting.

5 Grading (100 points as given below)

NOTE: The project is a team effort, all team members have to contribute to the project (brainstorm, program, debug). You could meet face-to-face in school/elsewhere or Skype offers "sharing screen" feature, which might be useful if you want to show each others code while talking remotely. During assessment, you will be asked about individual contributions to the project and may also be quizzed. It is likely that if you are unable to answer the questions, then your grade will suffer, although your project met all expectations. If a project member has not contributed to the project, then he/she may see a grade penalization.

1. Stage 1: 40 points (week of March 18 - 22). Please email course TA, Andrew Waterreus, ajwhersh@gmail.com to set up a time to meet up (all team members should be present). Please schedule at least 15 min.

2. Stage 2: 50 points (week of April 1 - 5). Please email course instructor, Pranav Bhounsule, pranav.bhounsule@utsa.edu to set up a time (all team members should be present). Please schedule at least 15 min.

3. Stage 2: 10 points (by end of the the day April 5, 2019). Email your game to the instructor Pranav Bhounsule, pranav.bhounsule@utsa.edu. Please put your name on the x-axis using xlabel("name1 and name2");

4. Penalties: 10 points each for missing deadline for Stage 1 or Stage 2.