CS 2261 Lab 04: States, Structs, and Pooling

Provided Files

- `main.c`
- `myLib.c`
- `myLib.h`
- `game.c`
- `game.h`

Files to Edit/Add

- `main.c`
- `game.c`
- `game.h`
- `Your Makefile`

Instructions

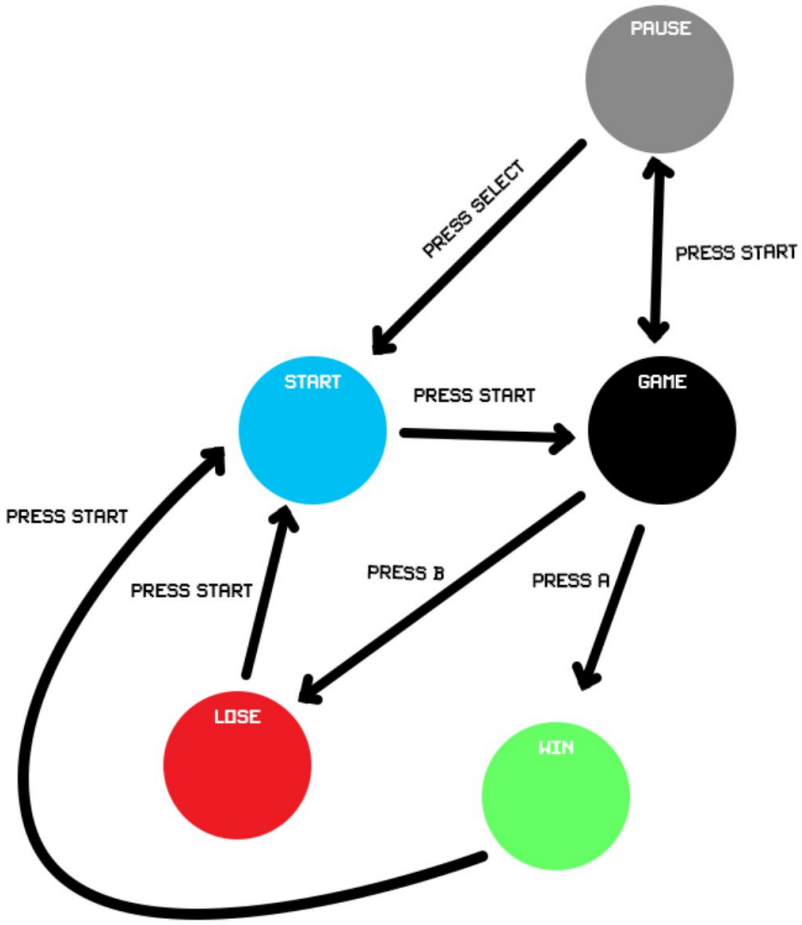
In this lab, you will be completing several different TODOs, which will, piece by piece, make a simple Space-Invaders-like game. Each TODO represents a component of the game, and is broken down into sub-TODOs. Your code will likely not compile until you complete an entire TODO block, at which point the game should compile with the new component added and working.

After you download and unzip the files, add your Makefile, add the SOURCES with it, and then compile and run it. At this point, you should see just a blank black screen. Complete the TODOs on order, paying close attention to the instructions.

TODO 1 – State Machine

- For our game to be user-friendly, we need to implement a state machine first. The state machine for this assignment will have the following states:
    - START

        - Consists of a blank cyan screen
        - The seed for the random number generator increases each frame spent in this state
        - Pressing START takes you to the GAME state, seeds the random number generator, and calls initGame()
    - GAME
        - Consists of a blank black background with a game running on top of it (we will add in the game in later TODOs)
        - Calls updateGame(), waitForVBlank(), and drawGame()
        - Pressing START takes you to the PAUSE state
        - Pressing A takes you to the WIN state (for now)
        - Pressing B takes you to the LOSE state
    - PAUSE
        - Consists of a blank gray screen
        - Pressing START takes you to the GAME state without reinitializing the game
        - Pressing SELECT takes you back to the START state
    - WIN
        - Consists of a blank green screen
        - Pressing START takes you back to the START state
    - LOSE
        - Consists of a blank red screen
        - Pressing START takes you back to the START state
- To help you visualize, the state machine looks like this:



- TODO 1.0: In the main while loop, create the switch-case for the state machine.
- TODO 1.1: Below the already-written initialize function, make the functions for all of your states with their respective goTo functions. Make sure that the states work like the ones described above, and that you don't do things every frame that only need to be done once (like fillScreen)
- TODO 1.2: Now that you have made these functions, put the prototypes for them at the top of main.c.
- TODO 1.3: Call your goToStart function in the initialize function so that when the game starts, the START state is first. If you haven't already done so, call your state functions in the state machine switch-case.
- Compile and run. You should be able to travel through all the states by pressing the respective buttons. If not, fix this before going further.

TODO 2 - Player

- Now that we have our state machine set up, we can begin adding in the game. We will do this in game.c and game.h, and call those functions in main.c (like we already are with initGame(), updateGame(), anddrawGame()). Let's start by getting the player working.
- TODO 2.0: In game.h, create the struct for the player, typedefed PLAYER. The player should have a row, col, oldRow, oldCol, cdel, height, width, color, and a bulletTimer (we will see the use for this later).
- UNCOMMENT 2.0: Below this, uncomment the line that declares the player.
- UNCOMMENT 2.1: In game.c, uncomment the line that declares the player here.
- UNCOMMENT 2.2: Below the drawBar function, uncomment initPlayer(), updatePlayer(), and drawPlayer().
- UNCOMMENT 2.3: Now that we have these player functions, uncomment the prototypes in game.h
- UNCOMMENT 2.4: Uncomment initPlayer() in initGame()
- UNCOMMENT 2.5: Uncomment updatePlayer() in updateGame()
- UNCOMMENT 2.6: Uncomment drawPlayer() and drawBar() in drawGame()
- Compile and run. When you enter the game state, you should see the player under the red bar, and move the player left and right. If not, fix this before going further.

TODO 3 - Bullets

- Now that the player is moving, let's give it a pool of bullets to shoot.
- UNCOMMENT 3.0: In game.c, uncomment the line that declares the pool of bullets here.
- UNCOMMENT 3.1: Below the drawPlayer function, uncomment initBullet(), fireBullet(), updateBullet(), and drawBullet().
- TODO 3.0: Complete the initBullets() by initializing all the bullets with the following values:
    - height – 2
    - width – 1
    - row – negative the bullet's height
    - col – 0
    - oldRow – the player's row
    - oldCol – the player's col
    - rdel – negative 2
    - color – white
    - active – 0

- TODO 3.1: Complete the updateBullets() function. This takes in a pointer to a bullet. If the bullet is inactive, the function does nothing. If it is active, it moves the bullet up the screen. If the bullet goes off the top of the screen, make it inactive.
- UNCOMMENT 3.2: Now that we have these bullet functions, uncomment the prototypes in game.h
- UNCOMMENT 3.3: Uncomment initBullets() in initGame()
- TODO 3.2: In updateGame(), call updateBullet() for each of your bullets.
- TODO 3.3: Complete fireBullet(). This should iterate through the bullets to find the first inactive bullet in the pool and initialize it. When you are finished initializing, break out of the loop. Initialize the bullet like so:
    - row – the player's row
    - col – player col + (player width / 2) + (bullet width / 2)
        - This is the center of the player's top
    - active – 1
    - erased – 0
- UNCOMMENT 3.4: In updatePlayer(), uncomment fireBullet() so that they actually fire
- TODO 3.4: Since pressing A fires a bullet, it should not also win the game every time you press it. So, back in main.c, comment out the fact that pressing A wins the game.
- TODO 3.5: In drawGame(), call drawBullet() for all of the bullets.
- Compile and run. When you enter the game state, you should be able to fire bullets by pressing A. You should be able to see three at once if you fire quickly enough. If not, fix this before going further

TODO 4 - Balls

- Now that the player is moving and shooting, let's give it them something to shoot at.
- TODO 4.0: Most of the code to make the balls work has already been written for you, so in updateGame(), call updateBall() for each of your balls.
- TODO 4.1: In drawGame(), call drawBall() for all of the balls.
- Compile and run. When you enter the game state, you should be able to see several balls bouncing around. When you pause and unpause, they shouldn't have moved. If not, fix this before going further. Every time you re-run the game, the starting positions of the balls should be the same. If not, you aren't using srand() correctly.

TODO 5 - Collision

- Having all those balls bounce around isn't very fun yet. Make it so that shooting all of them allows you to win the game.
- TODO 5.0: In updateBall(), loop through all of the bullets. If an active bullet is colliding with this ball, make both the bullet and the ball inactive, then decrement ballsRemaining.
- TODO 5.1: In order for the player to be able to win the game, you need to go back into main.c, and, in the game state function, transition to the WIN state if ballsRemaining is 0.
- Compile and run. When you enter the game state, you should be able to shoot the balls. If a bullet hits a ball, they both disappear. If you shoot all the balls, the win state should begin. If not, fix this.

At this point, you should be able to travel to all of the states, play the game, and win the game. If so, then you are done.

Submission Instructions

Zip up your entire project folder, including all source files, the Makefile, and everything produced during compilation (including the .gba file). Submit this zip on Canvas. Name your submission Lab04_FirstameLastname, for example:"Lab04_LoppakSlusk.zip".