Game Console (Report)

By Henry Earnest, Michael Rostom, and Nathan Kim

1. A short introduction giving an overview of your project and what assumptions you are making about the user.
    a. We created a retro game console capable of running two arcade-inspired games at exceptionally high frame rates on an SPI TFT screen. Important features include optimized SPI drawing, UART transmission of high scores to a host computer, ISRs for switching and restarting games, and more. The users of the console are expected to be comfortable looking at a small screen, and using small devices like buttons and joysticks with their hands. Users also should be comfortable with playing games.

Requirements

Console Operation Requirements

RO-0: Console shall start in the Menu.

RO-1: If console is in the Menu:

A. A menu displaying two game choices shall be visible.
B. Joystick input to the left shall launch Game 1.
C. Joystick input to the right shall launch Game 2.
D. If the user presses any button, nothing shall happen.

RO-2: If console is in any Game:

A. If the user presses the “Home” button, the Menu shall be displayed as if the Arduino restarted, and all game data besides scores shall be reset.
B. If the user presses the “Restart” button, the current game played shall restart from its beginning state, and all game data besides scores shall be reset.

RO-3: Score shall be retrieved from a computer and displayed at the start of each game.

RO-4: High score shall be sent back to a computer at the end of each game.

Ping (Game 1) Requirements

Parameters:

A. The playable game grid is 320x240 pixels
B. The paddle is 8 x 64 pixels
C. The ball is 16 x 16 pixels
D. The ball's gravity is 40 px/s^2
E. The possible gravity directions are: LEFT, RIGHT.
F. Gravity shall increase by 9 px/s^2
G. Bounciness shall be 10% (0.1)

G1-RO-1: At the start of the game:

A. The score shall be initialized to 0
B. The ball speed shall be initialized at 0.
C. Gravity shall be initialized to 40; Gravity Direction shall be initialized to LEFT
D. The paddle position shall start at (x = 0, y=120)
E. The ball shall start at position (x = 120, y = 160)

G1-RO-2: If the ball is moving, the ball shall accelerate according to the current Gravity and Gravity Direction.

G1-RO-3: If the ball collides with a wall or paddle:

A. If the ball is colliding with the right wall, the ball shall negate its x velocity and randomize the y velocity
B. If the ball is colliding with the top or bottom wall, the ball shall bounce at the reflection angle (negate its y velocity).
C. If the ball is colliding with the paddle, the ball shall bounce and increment the score by 1
D. Bounces on different locations the paddle shall result in different angle variations.
E. If the ball is colliding with the left wall, ***the game shall be over.***

G1-RO-4: The paddle shall move up or down if upward or downward joystick input is received, proportional to the upward or downward input's magnitude, unless the paddle would overlap with the top or bottom of the screen.

G1-RO-5: Player score shall increase every time the ball hits the paddle.

Dodge (Game 2) Requirements

Parameters:

- The playable game grid is 320x240 pixels

- The player is 16 x 16 pixels
- Each bullet is 12 x 12 pixels
    a. The possible Bullet Directions are: UP, DOWN, LEFT, RIGHT.
    b. Bullets move at a certain speed, as detailed in G2-RO-1

G2-RO-1: At the start of the game:

A. The score shall be initialized to 0
B. Player position shall be at the middle of screen: 160x120.
C. Bullet speed begins at 150 pixels per second.
D. Bullet speed increases to 300 pixels per second at a rate of $2px/s^2$.
E. Bullet spawn interval is 250ms.
F. Bullet spawn interval decrease is .005ms/s.

G2-RO-2: If the player is at a move step:

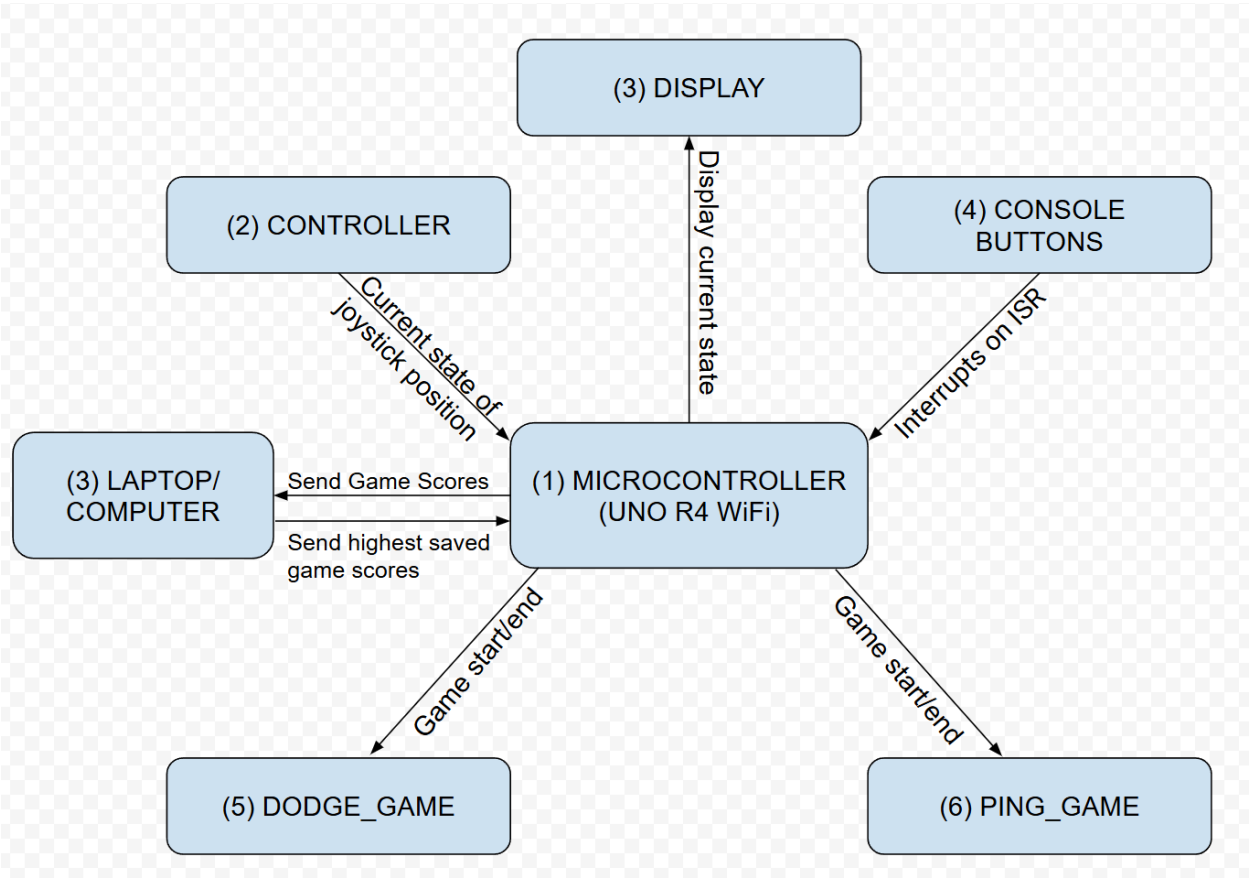
A. Player shall move in space according to Joystick input proportional to the input's magnitude.
B. All bullets on screen shall move according to their direction and location.
C. If there is a collision between a bullet and the player, ***the game shall be over.***

G2-RO-3: When a bullet is spawned, it shall be positioned randomly at an edge of the screen and move towards the opposite edge.

G2-RO-4: When a bullet leaves the screen, it is deleted (rendered inactive)

Architecture Diagram

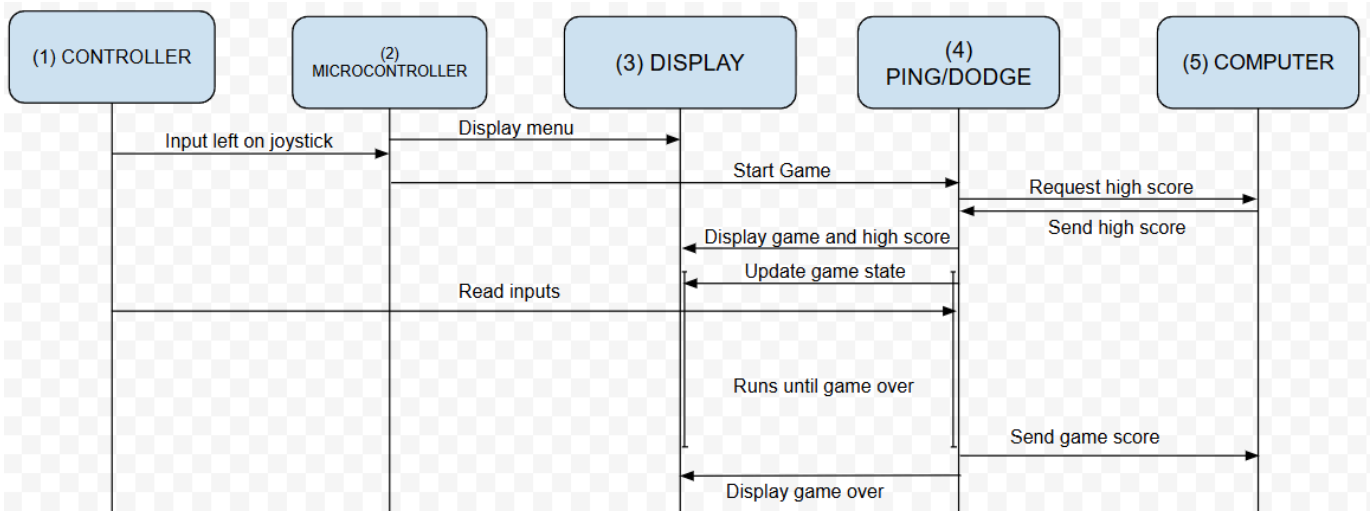
2. *Revised architecture diagram from part 2 of the milestone report.



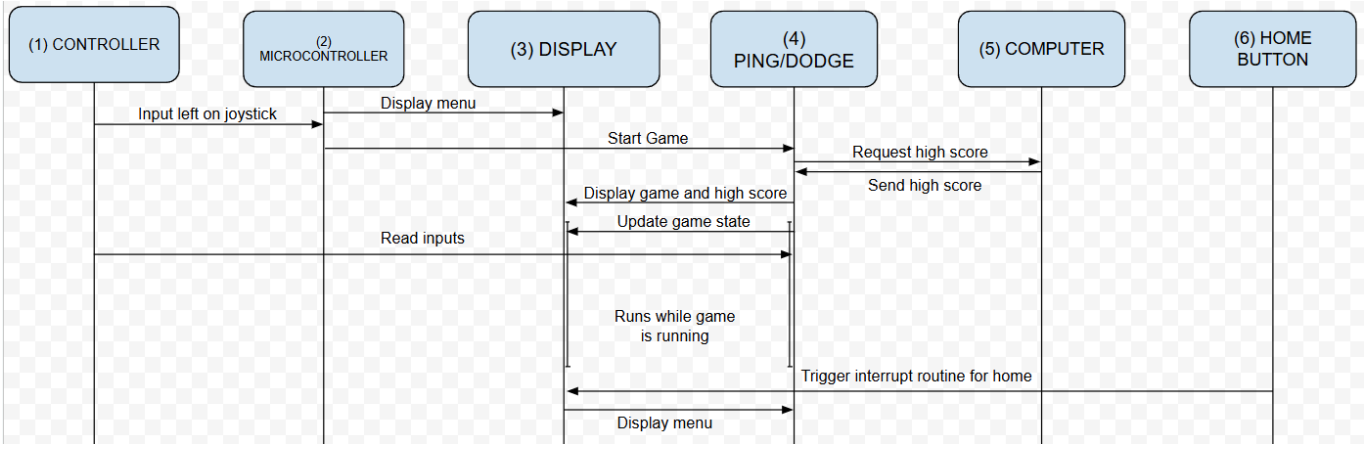
Sequence Diagrams

3. Three sequence diagrams for reasonable use case scenarios of your system (if there are more than 3, you should pick the 3 most likely use case scenarios to diagram, and include a note as to what scenarios you left out). Before *each diagram, you should write a short (1-2 sentence) description of what the use case scenario is.

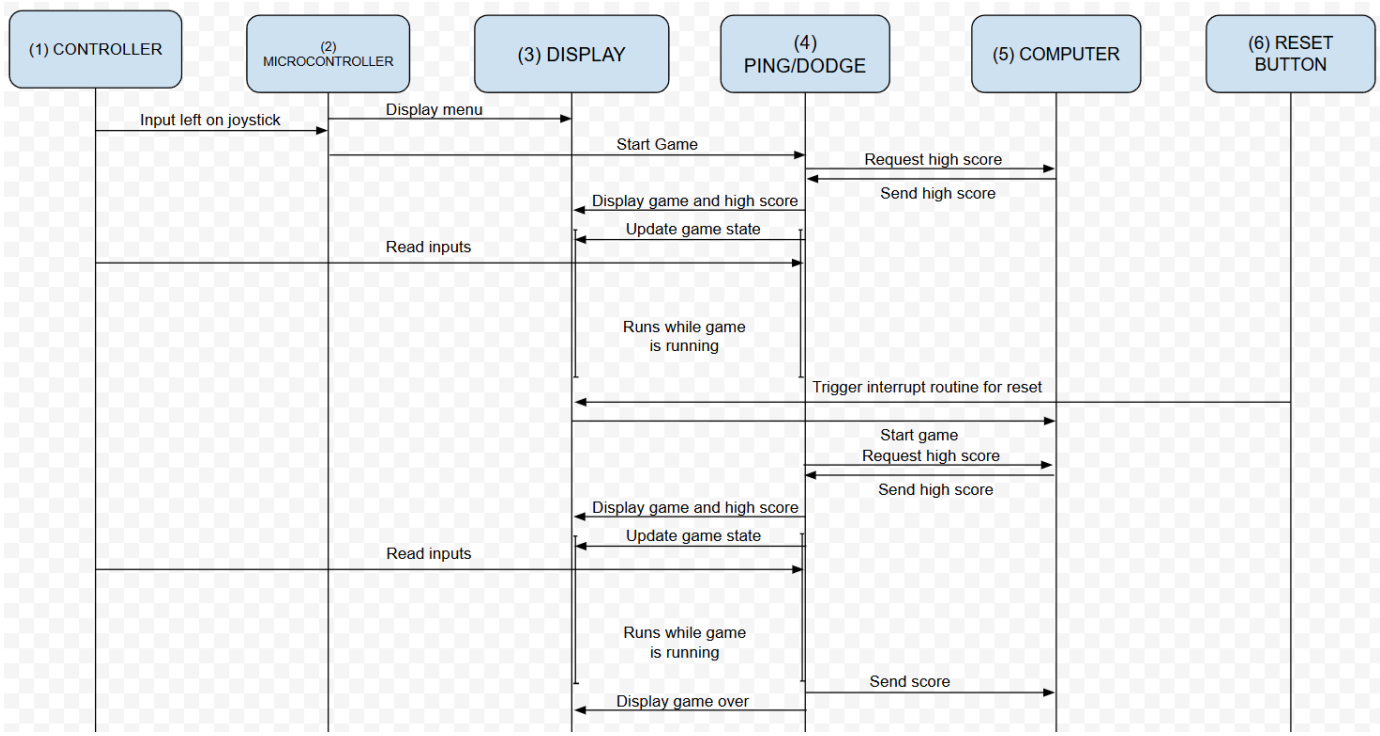
In this sequence diagram, this is a case where a player starts a game of ping or dodge and loses a game of ping or dodge. Since ping and dodge are roughly similar when in the sequence diagram they are interchangeable.



In this sequence diagram, this case is when a player starts a game of ping/dodge and proceeds to hit the home button to return to the menu.

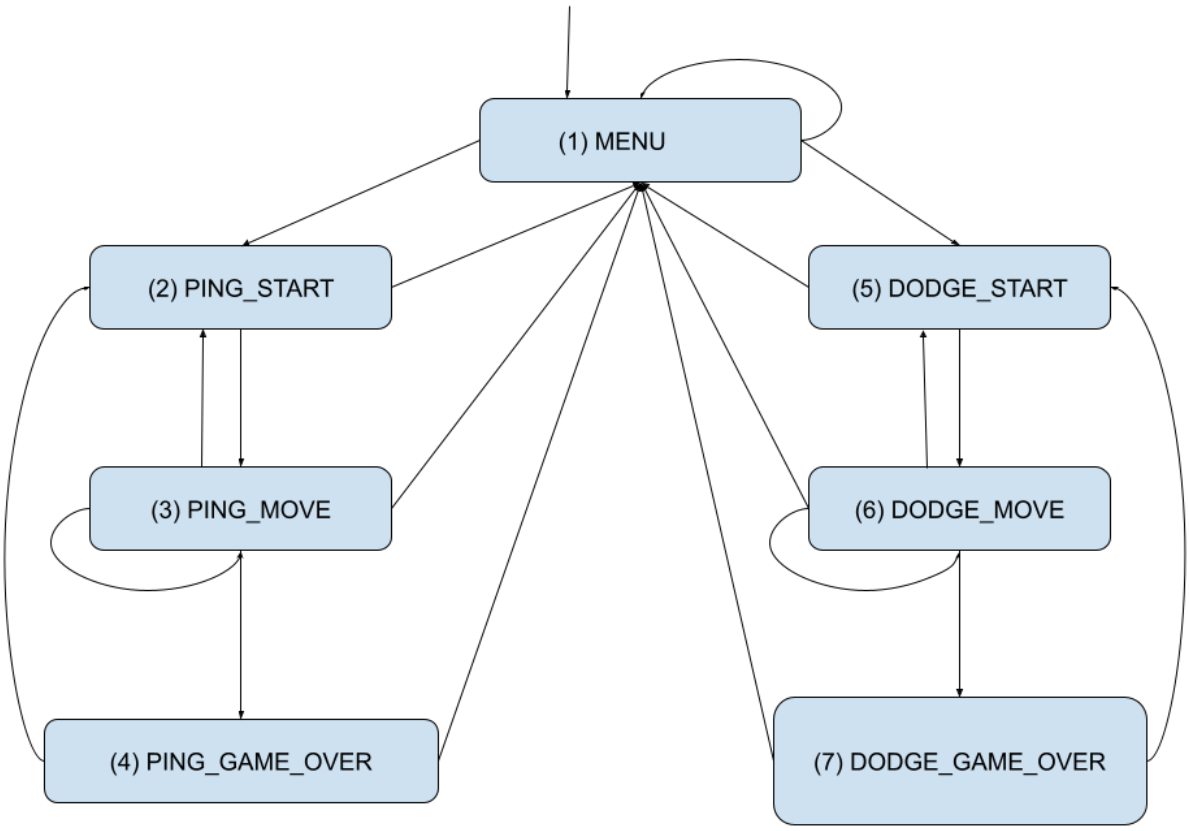


In this sequence diagram, this case is where a player starts a game of ping/dodge and hits the reset button which restarts the game from the beginning. The player then plays until the game ends.


(1) CONTROLLER
(2) MICROCONTROLLER
(3) DISPLAY
(4) PING/DODGE
(5) COMPUTER
(6) RESET BUTTON
Input left on joystick
Display menu
Start Game
Request high score
Send high score
Display game and high score
Update game state
Read inputs
Runs while game is running
Trigger interrupt routine for reset
Start game
Request high score
Send high score
Display game and high score
Update game state
Read inputs
Runs while game is running
Send score
Display game over

Revised Extended State Machine

4. *Revised finite and/or extended state machine from part 4 of the milestone report. See part 13 below for directions on including your peer review feedback as an appendix.



Inputs	**Description**
Joystick_input	*Struct representing x and y joystick input*
ResetButtonPressed	*Boolean representing whether reset ISR state is active due to reset button being pressed*
HomeButtonPressed	*Boolean representing whether home ISR state is active due to home button being pressed*

Outputs	**Description**
drawGameOver()	*Draws the game over screen for Dodge or Ping*

drawMenu()	*Draws the main menu*
DrawPingIntro()	*Draws the intro to Ping, showing the high score*
drawBackWall()	*Draws the back wall of Ping*
updateBall()	*Moves the ball in Ping according to its current gravity and direction*
updatePaddle()	*Moves the paddle in Ping according to joystick y input*
collideBall()	*Checks for ball collisions in Ping, changing ball direction as needed, and incrementing score on paddle collisions*
drawBall()	*Draws the ball in Ping at its current location*
drawPaddle()	*Draws the paddle in Ping at its current location*
DrawDodgeIntro()	*Draws the intro to Dodge, showing the high score*
spawnBullet()	*Spawns a bullet at a random location along a random edge of the map, setting its velocity to move towards the opposite edge.*
updatePlayer()	*Moves the player proportional to current joystick input*
drawPlayer()	*Draws the player at its current location*
updateBullets()	*Moves the bullets based on their speed and direction*
checkBulletCollisions()	*Checks for collisions between the player and bullets, which would set player's alive field to false. Also deletes bullets that leave the screen*
processResponse()	*Processes response from computer through UART*
sendNewScore()	*Sends new score of a game from Arduino to computer through UART*

Variables	**Description**
Ball	*Struct representing properties of ball in ping: x, y, dx, dy, and bounciness*
player	*Struct representing properties of player in dodge: x, y, and alive*

Transition	**Guard**	**Explanation**	**Output**	**Variables**

1-2	joystickInput.x > 250 and HomeButtonPressed = false	If the joystick is moved right, user has chosen Ping		
1-5	joystickInput.x < -250 and HomeButtonPressed = false	If the joystick is moved left, user has chosen Dodge		
2-3	HomeButtonPressed = false	Display the Intro screen before the game.	ReceiveScore() DrawPingIntro() drawBackWall() DrawPaddle() drawBall()	Ball.x = 120 Ball.y = 160 Ball.dx = 0 Ball.dy = 0 Ball.bounciness = 0.1
3-3	Ball.x > 0 and ResetButtonPressed = false and HomeButtonPressed = false	As long as ball is on screen, move ball	updatePaddle() updateBall() collideBall() drawBall() drawPaddle()	
3-4	Ball.x <= 0 and ResetButtonPressed = false and HomeButtonPressed = false	Ball went off screen, player lost	sendNewScore() displayGameOver()	
4-2	ResetButtonPressed = true and HomeButtonPressed = false	If the reset button is pressed, reset the game.		ResetButtonPressed = false
3-2	ResetButtonPressed = true and HomeButtonPressed = false	If the reset button is pressed, reset the game.		ResetButtonPressed = false
5-6	HomeButtonPressed = false	Display the Intro screen before the game.	ReceiveScore() DrawDodgeIntro() drawPlayer()	Player.x = screenWidth/2 Player.y = screenHeight/2 player.alive = true
6-6	Player.alive = true and ResetButtonPressed = false and HomeButtonPressed = false	As long as player is alive, spawn bullets and move them and player	spawnBullet() updatePlayer() drawPlayer() updateBullets() checkBulletCollisions()	

6-7	Player.alive = false and ResetButtonPressed = false and HomeButtonPressed = false	Player is dead	sendNewScore() displayGameOver()	
7-5	ResetButtonPressed = true and HomeButtonPressed = false	If the reset button is pressed, reset the game.		ResetButtonPressed = false
6-5	ResetButtonPressed = true and HomeButtonPressed = false	If the reset button is pressed, reset the game.		ResetButtonPressed = false
7-1	HomeButtonPressed = true	If the home button is pressed, go back to menu	drawMenu()	
6-1	HomeButtonPressed = true	If the home button is pressed, go back to menu	drawMenu()	
5-1	HomeButtonPressed = true	If the home button is pressed, go back to menu	drawMenu()	
2-1	HomeButtonPressed = true	If the home button is pressed, go back to menu	drawMenu()	
3-1	HomeButtonPressed = true	If the home button is pressed, go back to menu	drawMenu()	
4-1	HomeButtonPressed = true	If the home button is pressed, go back to menu	drawMenu()	
1-1	joystickInput.x > -250 and joystickInput.x < 250	If the user hasn't pressed		

		anything, stay in menu		

5. A discussion of what environment process(es) you would have to model so that you could compose your FSM with the models of the environment to create a closed system for analysis. Composing state machines was covered during the modeling lectures. For each environmental process, you should explain and justify whether it makes sense to model it as either a discrete or hybrid system and either a deterministic or non-deterministic system (for example, a button push could be modeled as a discrete, non-deterministic signal because it is an event that could happen at an arbitrary time, but a component's position could be modeled as a hybrid, deterministic signal based on some acceleration command). You are not being asked to actually do this modeling or any sort of reachability analysis, just to reason about this sort of modeling at a high level.

    a. Reset Button: modeled as a discrete, non-deterministic signal because it is an event that can only take on one of two values, and that could happen at an arbitrary time.
    b. Home Button: same reasoning; modeled as a discrete, non-deterministic signal because it is an event that can only take on one of two values, and that could happen at an arbitrary time.
    c. Joystick: modeled as a hybrid, non-deterministic signal based on some acceleration command, because the input is analog along multiple ranges and can change at an arbitrary time and rate.

Revised Traceability Matrix

6. *Revised traceability matrix from part 5 of the milestone report.

The traceability matrix was completed here: Traceability Matrices

Testing Approach

7. *An overview of your testing approach, including which modules you unit tested and which integration and system (software and user-facing/acceptance) tests you ran. You should explain how you determined how much testing was enough (i.e. what coverage or other criteria you used, how you measured that you achieved that criteria, and why you think it is sufficient for you to have met that criteria).

For our testing approach, we unit tested the following modules:

- Menu
- Ping
- Dodge
- Button ISRs

Non-unit tests run on each module:

1. Menu
    - Acceptance testing:
        - The screen displays the correct text on program launch
    - Integration tests:
        - Moving the joystick left successfully switches to Dodge, and Dodge runs correctly
        - Moving the joystick right successfully switches to Ping, and Ping runs correctly
        - The menu is correctly drawn when the home button is pressed
2. Ping
    - Acceptance testing:
        - The player can move the paddle vertically with the joystick
        - The ball visibly bounces when hitting the paddle and walls
        - The game over sequence and screen is played when the ball hits the left wall
    - Acceptance testing (from demo playtests):
        - A variety of players can reach moderate scores when playing the game
        - Players can reach high scores when playing the game, indicating a good gameplay balance of difficulty over time
            - High score reached: 155
            - Typical score: 8
    - Integration testing:
        - The home button returns to the menu when pressed during Ping
        - The restart button restarts Ping
3. Dodge
    - Acceptance testing:
        - Bullets visibly appear during gameplay
        - Running into a bullet causes a game over sequence to happen
        - The bullet spawn rate visibly increases over time
    - Acceptance testing (from demo playtests):
        - A variety of players can play the game successfully, reaching scores of

50+
        - Players can reach high scores of 200+, showing good gameplay balance of bullet spawn rate increase over time
            - High score reached: 225
            - Typical score: 30
    - Integration testing:
        - The home button returns to the menu when pressed during Dodge
        - The restart button restarts Dodge
4. Watchdog
    - Acceptance testing:
        - adjusting `delay()` values in the main loop to make the watchdog bark. This was done throughout the development process at different times, ensuring that it still worked despite code changes.
        - Playing the game after tuning the watchdog, ensuring it doesn't bark
    - Integration testing:
        - The watchdog doesn't bark during longer delays caused by cutscenes (achieved through petting the watchdog between cutscene delays)
5. Button ISRs
    - Acceptance testing:
        - logging when homeISR and resetISR are triggered by button press
        - visibly confirming that Dodge is restarted when Dodge is being played and the restart button is pressed
        - visibly confirming that Ping is restarted when Ping is being played and the restart button is pressed
        - Visibly confirming that the home button causes the menu to be displayed
    - Integration testing:
        - Pressing the home button returns the FSM to the MENU state
        - Pressing the restart button returns the FSM to the START state of the current game

To determine how much testing was enough, we created a list of every function that had testable functionality, and wrote thorough tests for each of these functions. To determine what was "thorough" in terms of testing a given function, we looked at the major cases of what the function can accomplish (ex: collideBall() having 4 unique bounce cases) and tested those for accuracy. We decided that this criteria was sufficient because those major cases of gameplay functions form the entirety of the gameplay; a playthrough of our games was fully covered by our tests.

On top of this, when writing tests, we took into account the design artifacts we produced during our design phase. Some tests specifically tested the extended state machine that we produced during the design phase, making sure that certain transitions occurred as expected. Other tests ensured that certain requirements were fulfilled by related functions by testing the correctness of those functions. We performed acceptance tests in accordance with our sequence diagrams, making sure that each step of the diagram can be successfully followed.

Safety and Liveness Requirements

8. At least 2 safety and 3 liveness requirements, written in propositional or linear temporal logic, for your FSM. Include a description in prose of what the requirement represents. Each requirement should be a single logic statement, made up of propositional and/or linear temporal logic operators, that references only the inputs, outputs, variables, and states defined in part 5 of this report. We will discuss safety and liveness requirements on the week of November 27th. You can also refer to chapter 13 of Lee/Seshia or chapters 3 and 9 of Alur's textbook (Brown login required).

Safety:

- G(player.x >= 0 and player.y >= 0 and player.x <= screenWidth - playerSize and player.y <= screenHeight - playerSize)

Globally, the player object must always be on screen and can never go off screen from any edge.

- G(¬(DODGE_GAME_OVER and Player.alive == true))

Globally, the dodge game shall not be over and the player is alive.

Liveness:

- G(HomeButtonPressed => F(drawMenu))

Whenever the home button is pressed, we should eventually draw the menu for the user to choose a game.

- Joystick_input.x < -250 => F(player.x = screenWidth / 2 and player.y = screenHeight / 2 and player.alive = true)

If there is a left input into the joystick on the menu, eventually the player will spawn in the middle of the screen.

- G(PING_GAME_OVER and HomeButtonPressed = > F(MENU))

If the FSM is in state PING_GAME_OVER and the home button is pressed, then eventually we reach the MENU state in the FSM.

Codebase Description

9. A description of the files you turned in for the code deliverable. Each file should have a high-level (1-3 sentence summary) description. For each of the assignment requirements (PWM, interrupts, serial, etc), you should indicate which file(s) and line(s) of code fulfill that requirement.

Requirements:

- ADC: GameConsole.cpp (Function pollInputs(), lines 26-35)
- WDT:
    - watchdog_utils.cpp (Entire file)
    - Pet functions:
        - GameConsole.cpp (Function updateFSM, lines 41-70)
        - Dodge_utils.cpp (Function PingUpdateFSM, lines 120-191)
        - Ping_utils.cpp (Function DodgeUpdateFSM, lines 192-268)
- ISR:
    - isr_utils.cpp (Entire file)
- Serial Communication:
    - uart_utils.cpp (Entire file)
    - UART-Laptop.py (Entire file)

Descriptions:

- Dodge
    - Dodge_utils.cpp
        - This file holds all the logic required to run dodge and consists of the FSM needed to run the game correctly.
    - Dodge_utils.h
        - This file holds the header file for dodge_utils which contains any functions that might need to be publicly defined for other functions to use.
- Ping
    - Ping_utils.cpp
        - This file holds all the logic required to run ping and consists of the FSM needed to run the game correctly.
    - Ping_utils.h
        - This file holds the header file for ping_utils which contains any functions that might need to be publicly defined for other functions to use.
- Utils
    - ISR
        - isr_utils.cpp
            - This file contains the functions needed to set up and run our ISR routines from the 2 buttons we have to pause and restart our game.
        - isr_utils.h
            - This file contains the header file which contains any functions that need to be publicly defined/called by other files for the ISR.

    - UART
        - uart_utils.cpp
            - This file contains the UART communication that we use between the arduino and computer. It has 2 functions which can be used to send a request and receive a request.
        - uart_utils.h
            - This file contains the header file which contains functions that need to be publicly called by other functions to access UART communications
        - UART-Laptop.py
            - This is the host python file that needs to be run to save scores to the computer that is hosting the game. This is to ensure that scores are saved even if the arduino is unplugged.
            - Note: Requires pyserial library
    - Watchdog
        - watchdog_utils.cpp
            - Contains functions needed to set up and pet the watchdog when required.
        - watchdog_utils.h
            - Contains the header file which defines and functions that need to be accessed by other files to setup or pet the watchdog.
- GameConsole
    - GameConsole.cpp
        - This file contains the main code that runs when the device is turned on and contains the controller and menu logic for the game console.
    - GameConsole.h
        - Contains links to all the separate header files and contains any functions needed to run the menu and game overall.

10. A procedure on how to run your unit tests (for example, if you have mock functions that are used by setting a macro, similar to lab 6, make sure to note that).
    a. Uncomment `#define UNIT_TEST` in GameConsole.h to

Reflection

11. A reflection on whether your goals were met and what challenges you encountered to meet these goals. You should only include challenges met or newly addressed since the milestone.

Since the milestone,

- We replaced the gfx library from Adafruit-GFX-Library to Arduino_GFX. Which helped us with problems with software SPI and heavily sped up the screen refresh rate.
- We fixed the anticipated problem “too many bullets in dodge causing Arduino to slow down” by optimising the drawing of bullets and player. We also lowered the spawn rate and max number of bullets according to the play tester’s feedback.
- We met a new challenge of moving objects flickering to a distracting amount. We were able to overcome this by only keeping track of the old and new objects and draw and erasing the difference between the two.
- Another challenge was coding in VS-Code instead of the Arduino IDE. Which required us to try different extensions and workflows until we settled on PlatformIO.
- We faced a challenge with merging code from different contributors. Since each person had different coding styles and ideas of how to do things. We often had to rewrite code sections to merge them together.

Appendix: Review Spreadsheets

12. As an appendix, include the review spreadsheets you received after the milestone demo. Each defect should be marked as “fixed” or “will not fix” with a justification.

https://docs.google.com/spreadsheets/d/14IYSH-s3QGx_APgqxPaT8W5XVSSwQVsxGWhcyqsBIww/edit?gid=0#gid=0

https://docs.google.com/spreadsheets/d/1kXa99LOSGMlSTmfplGTl1CprfA0Az-ow1z5XGbSBZfO/edit?usp=sharing

https://docs.google.com/spreadsheets/d/1ZJuYaxS5CQqu2zzeibbTyh13mKmqazMlgSLNCDi4eSQ/edit?usp=sharing

https://docs.google.com/spreadsheets/d/1Sr4TofEz9AZ5x-C3v85nh-Ca8Sj8dmJHXCsgzldjlfA/edit?usp=sharing

Transition or state #	**Defect**	**Fixed / Will not fix**
State 1	While we can infer that the starting state of the FSM is State 1 for all FSMs, it would be helpful to have that a bit more clearly marked, for example with an arrow pointing to it!	Fixed
Game 1 and 2 FSMs	It seems that both Game 1 and Game 2 FSMs are identical, at least from the shown charts. I'll assume that they differ in the kind of guard clauses / variables they change, but this makes me wonder if the names of the states should be changed.	Fixed
All states	Oh no! None of the states have any input variables or output variables! I think you have a great draft of the initial states that would be used by your system, but it seems that there's still some work left to do in terms of the triggers that move between different states	Fixed
All transitions	Oh no! None of the transitions have any guard clauses. By extension, I think it's important to make sure that all of the transition guard clauses are mutually exclusive!	Fixed
Main Board FSM	When either Game 1 or Game 2 end with "Game Over" state, do we need to come back to the Main Board FSM to display some sort of message? I think this could be an interesting idea to consider	Will not fix - user can return home at any time with buttons
Main Board FSM	For the main Board FSM, one of the questions I had was about the edges that go to "Start Game 1" and "Start Game 2". It seems like those are "end-point" states. Is that by design? This makes me think if it would be just possible to combine the Game 1 and Game 2 FSMs and the Board FSM into one thing that continues off of this "end-point" state?	Fixed - combined them
Behavior	General behavior and organization of your FSM seems to align closely with what you described in class. I think my overall feedback would be to go further with what you have as it's a great starting point!	Fixed - thanks

All	There are no listed inputs, outputs, or variables.	Fixed
All	Input and variable initialization is not included	Fixed
Move state (both)	is it possible for it to transition to itself? can you move and then move again before entering the collision state	Fixed
Check Inputs state (all)	why is this a separate state and not an input / variable for the states,	Fixed
Start Game 1/2	should these have a transition to the display screen state? From what the FSM looks like right now, once a game is over, you would need to rerun the entire game console to play again/play another game	Fixed
Collision (ping)	is there a distinction between the guards if the collision is with the back wall vs the paddle? i understand this is distinct from the game over condition (class with the wall behind paddle), but there are multiple possible surfaces to collide with and will the distinction happen in the FSM or in the code for the general collision state. Additionally, if collision can happen with the back wall, it doesn't make sense to transition to check inputs since it is not a collision with something we are inputting.	Fixed
	inputs, outputs and variables are not defined	Fixed
1	input and variable initialization is not included in the start up states, for all 3 FSMs	Fixed
	no guard conditions present on the FSM	Fixed
	variables and outputs aren't set in actions between state transitions	Fixed
	no idle state transitions present	Fixed
Outside of FSM	There are no inputs, outputs, or variables defined or initialized at all	Fixed.
Main Board, Game 1, Game 2 FSMs all transitions	There are no inputs, outputs, or variables. There are also no guard conditions and no actions. Therefore, inputs and variables are never used in any guard conditions, and outputs and variables are never specified/changed in the actions of the transitions, as required by the checklist.	Fixed.
Main Board FSM state 1, Game 1 FSM state 1, Game 2 FSM state 1	Each of these states are the state in the FSM that you start in. Usually these are notated with an empty transition that goes from nowhere into this state. This notation would be helpful, because currently the only way to know that these are the starting states are to read the state label.	Fixed.
Main Board, Game 1, Game 2	[nit] Not really a defect, but I notice you split this up into three FSMs. One for the main board and one for each of the two games. The FSMs are simple enough that I think it would be a bit more intuitive to combine them all as one FSM.	Fixed. Used feedback for improved FSM.
Main Board state 1	I don't think you need this state. You can just have an empty transition into state 2.	Fixed. Used feedback for improved FSM.

Main Board state 3	I don't think you need this state. You can just transition from state 2 to either state 4 or 5 depending on the user input. (need guard conditions for this)	Fixed. Used feedback for improved FSM.
Game 1 and Game 1, state 4	I'm not sure, but maybe you can do without this state? Just transition from state 3 to state 2, and insert the user input as an input in the state transition. Then state 2 can make the corresponding move depending on the input.	Fixed. Used feedback for improved FSM.