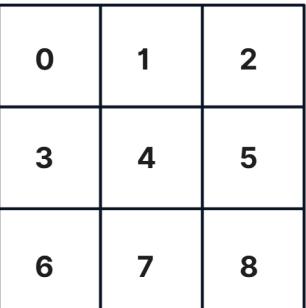
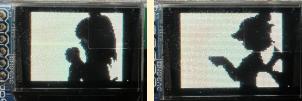


Group ID: S1\_03

Group Members: Ranee Ng Yin Yee (A0283075B), Nirala Tanishka Singh (A0282322M),  
Joel Chin Junwei (A0271839R), Sahej Agarwal (A0284719N)

Student	Improvement Description	Images/Photos
Team: Grid Masters	<p>Our Tic Tac Toe game has two modes.</p> <p><b>1V1 Mode (Player vs. Player)</b> Player 1 always plays as 'X', and Player 2 plays as 'O'. The game keeps track of player wins using a scoreboard displayed on the 7-segment display. Game ends once a player wins 3 rounds. Players can reset a round at any time before a win/draw occurs.</p> <p><b>1Vai Mode (Player vs. AI)</b> The AI always makes the first move and plays as 'O', while the player plays 'X'. The AI progresses through three levels (starting from level 1) of increasing difficulty, and the player must win the current level to advance to the next level. A timer on the 7-segment display shows how much time has elapsed in each round until a win, loss, or draw occurs for player's to take note of.</p>	<p>Grid box numbering system</p> 
Student A: Ranee  Improvement: UI, display, and state management	<p><b>FSM &amp; State Management</b></p> <ul style="list-style-type: none"> <li>Developed the finite state machine to manage transitions between display states</li> <li>Handled state changes based on user inputs and game data. Starts at <u>home state</u>. <i>From home state:</i> When btnC is pressed and sw == 0, switches to <u>mode selection state</u>. <i>From mode selection state:</i> When btnU/btnD is pressed, 1v1/vs AI selection is highlighted. When btnC is pressed and sw == 0, the respective mode is selected, switches to <u>1v1/ai state</u>. OR When btnL is pressed, switches to <u>home state</u>. <i>From 1v1/ai state:</i> When the game ends (win/loss/draw) switches to <u>game result state</u>. OR when btnL is pressed, switches to <u>mode selection state</u>. <i>From game result state:</i> After 5 (PVP) / 8 (AI) seconds, depending on score/level, switches to <u>1v1/ai state</u> or <u>home state</u>. For AI mode, level is incremented if game is won, level remains if game is drawn and level is set to 1 if game is lost.</li> <li>Reset and assign data appropriately during state transitions to maintain logic consistency.</li> </ul> <p><b>UI: Display Screens</b> Synchronized visuals across top and bottom OLEDs to reflect game state in real-time.</p> <ul style="list-style-type: none"> <li>Home Page Display starting screen</li> <li>Mode Selection Display</li> <li>Grid Display, Player Icon Display, Selection Display Rendered 3x3 game grid showing moves and highlighting selection.</li> <li>Turn Display indicate current player's turn</li> <li>Game Result Display win, loss, draw</li> </ul> <p><b>Game logic</b></p> <ul style="list-style-type: none"> <li>Implemented logic for selection highlighting system (display of selected but unconfirmed boxes)</li> </ul>	<p>Home page display</p>  <p>Mode selection display</p>  <p>Grid display, player icon display</p>  <p>Selection display</p>  <p>game result</p> 
Student B: Tanishka  Improvement: 1V1 game logic, win/draw logic, taking player input logic	<p><b>Taking Player Input &amp; Move Confirmation (1V1 and 1Vai mode)</b></p> <ul style="list-style-type: none"> <li>Use switches sw[0] to sw[8] to select one of the nine boxes (that is empty) on the grid. When a switch is turned on, the corresponding LED begins blinking and the corresponding grid box is highlighted to indicate the selected box.</li> <li>Press btnC to confirm the move. The blinking LED turns solid, confirming the player's move. <b>TAKE NOTE:</b> If you press btnC while multiple/no LEDs are blinking, the game will not progress. Please ensure to have only one LED blinking at any time while making your move.</li> </ul> <p><b>Turn Management</b></p> <ul style="list-style-type: none"> <li>Once the player's move is confirmed, the player's icon is rendered on the bottom OLED at the selected grid location.</li> <li>The system then automatically switches turns to the next player or the AI (indicated by the top OLED) continuing the game flow seamlessly.</li> </ul>	<p>Player Input, Move Confirmation, Turn Management</p>  <p>after btnC is pressed..</p> 

	<p><b>Win/Draw Feedback</b></p> <ul style="list-style-type: none"> <li>If <b>any player wins</b> (in 1v1 or 1Vai mode), LED0 to LED8 turn on in a running sequence to “celebrate”. If <b>AI wins</b> (in 1Vai mode), LED0 to LED8 start blinking. In both modes, if the round ends in a <b>draw</b>, LED[0:3] and LED[5:8] turn on in turns to indicate a tie. The top OLED shows the game result (X WON, O WON, DRAW) accordingly.</li> <li>After 5 seconds (for 1V1 mode), both the top and bottom OLEDs reset, and the game grid is cleared for the next round.</li> </ul> <p><b>Manual Reset of Round (1V1 Mode)</b></p> <ul style="list-style-type: none"> <li>Press btnR in 1v1 mode at any time before a win/draw is detected to reset the game grid and restart the round.</li> </ul>	<p><b>Win/Draw displays + led</b></p>  <p><b>1V1 mode reset display</b></p> 
<b>Student C:</b> Joel  <b>Improvement:</b> UI, scoreboard, round timer and animations	<p><b>1V1 Mode</b></p> <p><b>Game Information</b></p> <ul style="list-style-type: none"> <li>The scoreboard is displayed on the 7-segment display, tracking the number of wins for the players. (an0 = “o”, an1 = o_score, an2 = “X”, an3 = x_score)</li> <li>The top OLED displays the game mode and the current player's turn</li> </ul> <p><b>Final Winner Logic &amp; UI</b></p> <ul style="list-style-type: none"> <li>When either player reaches 3 wins, the game will end.</li> <li>An animation from Bad Apple plays on the top OLED for 3 seconds, depending on the player who won.</li> <li>When the animation finishes, the top OLED displays the overall winner for 5 seconds before returning to the menu.</li> </ul> <p><b>1Vai Mode</b></p> <p><b>Game Information</b></p> <ul style="list-style-type: none"> <li>A stopwatch timer is shown on the 7-segment display, tracking the game duration. It will pause and flash when the round is over.</li> <li>The top OLED displays the game mode and the current level of the game.</li> <li>A waiting animation plays while the AI processes the next move.</li> </ul> <p><b>Round Result UI</b></p> <ul style="list-style-type: none"> <li>When the round ends, an animation from Bad Apple plays on the top OLED for 3 seconds depending on the result (win, draw, lose).</li> <li>When the animation finishes, the top OLED displays the result of the game (You Win, Draw, You Lose) and the progression to the next stage (Return to L1, Restart Level, Move to Next Level) for 5 seconds before progressing.</li> </ul>	<p><b>Turn Indicator</b></p>  <p><b>Level Indicator + Wait Anim</b></p>  <p><b>Bad Apple Animations</b></p>  <p><b>Result Screens</b></p>  <p><b>Scoreboard</b></p>  <p><b>Round Timer</b></p> 
<b>Student D:</b> Sahej  <b>Improvement:</b> Onboard artificial intelligence, AI output on board	<p>Complete three levels of AI:</p> <ul style="list-style-type: none"> <li>Level 1: Random Moves, made fully randomized using the bouncing of the buttons</li> <li>Level 2: Medium Level AI: AI lacking a few abilities and being a bit short-sighted about the moves, allowing the user to win</li> <li>Level 3: Maximum Level AI: Undefeatable AI</li> </ul> <p>Once the AI has played its move, “O” is rendered on the grid where the AI makes a move and the respective led glows showing that the square is blocked.</p> <p>How it works and possible hints on how to beat:</p> <p>A full-depth minimax algorithm was initially implemented using logic gates; however, due to its extreme complexity (~9^109 modules and 60 register bits), it remained simulatable but not synthesizable. To support optimization, a Python script was developed to simulate optimal moves, and approximately 4500 KMap entries were manually analyzed to identify strategic patterns. Game moves were categorized into five key types: completing a three-in-a-row, blocking the opponent, creating a fork, blocking a fork, and selecting a fallback move. Early game logic prioritizes occupying the center or corners. The AI was refined into a medium-level version using finite state machines (FSMs), recursion, and sequential logic—resulting in a compact implementation that synthesized quickly and used only 0.75% of available LUTs.</p> <p>Three AI behaviors were integrated into a unified module. Additionally, input conflicts in Player vs AI mode were debugged to prevent overlap due to propagation delays, and visual responsiveness through LEDs was enhanced for clearer real-time feedback.</p>	

## Group ID: S1\_03

Group Members: Ranee Ng Yin Yee (A0283075B), Nirala Tanishka Singh (A0282322M),  
Joel Chin Junwei (A0271839R), Sahej Agarwal (A0284719N)

## References

### DeepSeek

- Ranee: used to improve the color matching algorithm of my python script for converting my pixel art to if else branches with colour assignments

The screenshot shows the DeepSeek interface. On the left, there is a code editor with Python code for color matching. On the right, there is a code editor with Verilog code. A tooltip provides information about solving a color overflow issue.

```
print("parameter PINK_HEX = 16#f899;")  
print("parameter BLUE_HEX = 16#177F;")  
print("parameter BG_BLUE_HEX = 16#000000;")  
print("parameter WHITE_HEX = 16#FFFF;")  
print(verilog_code)
```

To solve the color overflow issue in your image processing, we need to improve the color matching algorithm. The current method uses simple RGB855 distance, which can misclassify pixels at color boundaries. Here's an enhanced version with better color matching:

```
python  
from PIL import Image  
import numpy as np  
  
def find_consecutive_ranges(indices):  
    """Given a list of indices into consecutive ranges...  
    If not indices:  
        return []  
    indices = sorted(indices)  
    ...
```

### ChatGPT

- Joel: Used once to aid in the scoreboard logic, preventing it from incrementing repeatedly when the win signal is given.

The screenshot shows a conversation with ChatGPT. The user asks about preventing score increments. ChatGPT provides a Verilog code example and a common solution using a one-shot signal.

```
if (round_result == 2'b01) // X win  
    x_score <= x_score + 1;  
else if (round_result == 2'b10) // O win  
    o_score <= o_score + 1;  
This Verilog code increments x_score and o_score repeatedly, how can this be prevented?
```

Great observation! In Verilog, repeated or unintended increments usually happen when the conditions for updating the values are evaluated multiple times across clock cycles without proper control.

To prevent `x_score` and `o_score` from incrementing repeatedly, you need to ensure that the scoring logic only executes **once per event**, such as when a game round ends.

Common Solution: Use a One-Shot Signal or Flag

Introduce a control signal or flag (e.g., `score_updated`) that ensures the increment happens only once per round. Here's how you can do it:

### Picture2Pixel (<https://www.comp.nus.edu.sg/~guoyi/project/picture2pixel/>)

- Joel: Used to convert all of my UI graphics into Verilog code. Some graphics were manually optimised after conversion. The base script to convert pixel art into Verilog code for one frame and the Python script to combine multiple frames into a single block of code are used, with minor modifications to fit my uses.

```
command = [  
    'python', '-m', 'picture2pixel.convert2pixel',  
    image_path, str(width), str(height), str(svd_r), output_dir  
]  
  
subprocess.run(command)
```

### Pyxelate (<https://github.com/sedthh/pyxelate>)

- Joel: Used to convert the 720p frames of the Bad Apple animation into the resolution for use with Picture2Pixel.

```
# Convert image to pixel art  
image = io.imread(source_path)  
pyx = Pyx(height = 64)  
pyx.fit(image)  
new_image = pyx.transform(image)  
io.imwrite(image_path, new_image)
```

### Python Imaging Library (<https://omz-software.com/pythonista/docs/ios/PIL.html>)

- Joel: Used to further optimise the Bad Apple frames by bitmapping each frame into a 96x64 resolution image with only black and white pixels.

```
# Bad Apple Bitmapping Only
old_img = Image.open(image_path)
old_pixels = old_img.load()
new_img = Image.new('RGB', [96, 64], 'black')
new_pixels = new_img.load()
```

```
for j in range(new_img.size[0]):
    if j < 5 or j > 89:
        continue
    for k in range(new_img.size[1]):
        if (old_pixels[j - 5,k] > (128,128,128)):
            new_pixels[j,k] = (255, 255, 255)
        else:
            new_pixels[j,k] = (0, 0, 0)
new_img.save(bitmap_path)
```

Bad Apple!! (original video upload: <https://www.nicovideo.jp/watch/sm8628149>)

- Joel: This animation is commonly used in computing as a graphical and audio test. I used three 3-second-long snippets of this animation in the final project.



#### Course related feedback

NA