Project Proposal
“Lotus Arcade”

Team members:
Frigură-Iliasa Flavia-Mihaela - flavia-mihaela.frigura-iliasa@student.upt.ro
Libotean Ingrid-Silvia - ingrid-silvia.libotean@student.upt.ro
Moisi Andreea-Maria - andreea-maria.moisi@student.upt.ro
Popescu Carmen-Bianca - carmen-bianca.popescu@student.upt.ro
Talpeș Alexia-Monica - alexia-monica.talpes@student.upt.ro

Contact person:
Popescu Carmen-Bianca
0746153607
carmen-bianca.popescu@student.upt.ro

1) Project Description

This project involves the implementation of two classic games—**Tetris** and **Lotus Sweeper** (a reimagined version of Minesweeper)—on an **FPGA (Field Programmable Gate Array)** development board. The system will utilize **VGA** for video output and a **keyboard** for input. Upon starting the system, the user will be greeted with a simple **main menu** allowing game selection using the following keys:

- **T** – Launch Tetris
- **L** – Launch Lotus Sweeper
- **Q** – Quit application

The system will be built using **finite state machines (FSMs)** to manage game logic and a **VGA controller** to handle graphics rendering and user interface. The gameplay experience is designed to be engaging yet efficient in resource use, showcasing the FPGA’s real-time processing capabilities.

❖ Tetris

Tetris is a well-known puzzle game in which players must arrange falling geometric shapes—called **Tetrominoes**—within a vertical grid to form complete horizontal lines. The grid size will be **10 columns × 20 rows**, equivalent to a **100 × 200 pixel** area on the display. The Tetrominoes come in **5 distinct shapes**, each made up of four connected blocks.

Players can control the falling shapes using the arrow keys:

- **Left/Right Arrows** – Move shape left or right
- **Up Arrow** – Rotate shape
- **Down Arrow** – Increase falling speed

When a row is completely filled, it is **cleared**, and the player earns **points**. The game ends when the Tetrominoes stack up to reach the top of the screen. Additionally, the **falling speed increases** over time, increasing difficulty and requiring faster reaction times.

❖ **Lotus Sweeper**

Lotus Sweeper is a thematic twist on the traditional Minesweeper game. The player is presented with a grid of tiles, some of which conceal **lotus flowers (hazards)**. The objective is to uncover all the safe tiles—**lily pads**—without triggering a lotus.

Upon clicking a tile:

- If it's safe, it reveals either a blank tile or a **number** indicating how many adjacent tiles contain lotus flowers.
- If it's a lotus tile, the game ends.

Players can **mark suspicious tiles** using a flag-like indicator (referred to as **Leaves** in our version).

Controls:

- **Arrow Keys** – Move selection cursor
- **Enter/Space** – Reveal selected tile
- **F Key** – Place or remove a Leaf (flag)
- **L Key** – Toggle Lotus view mode (for debugging or extended gameplay mode)

Grid dimensions and gameplay settings (e.g., number of lotus tiles) are still under discussion and will be determined based on VGA resolution and memory constraints.

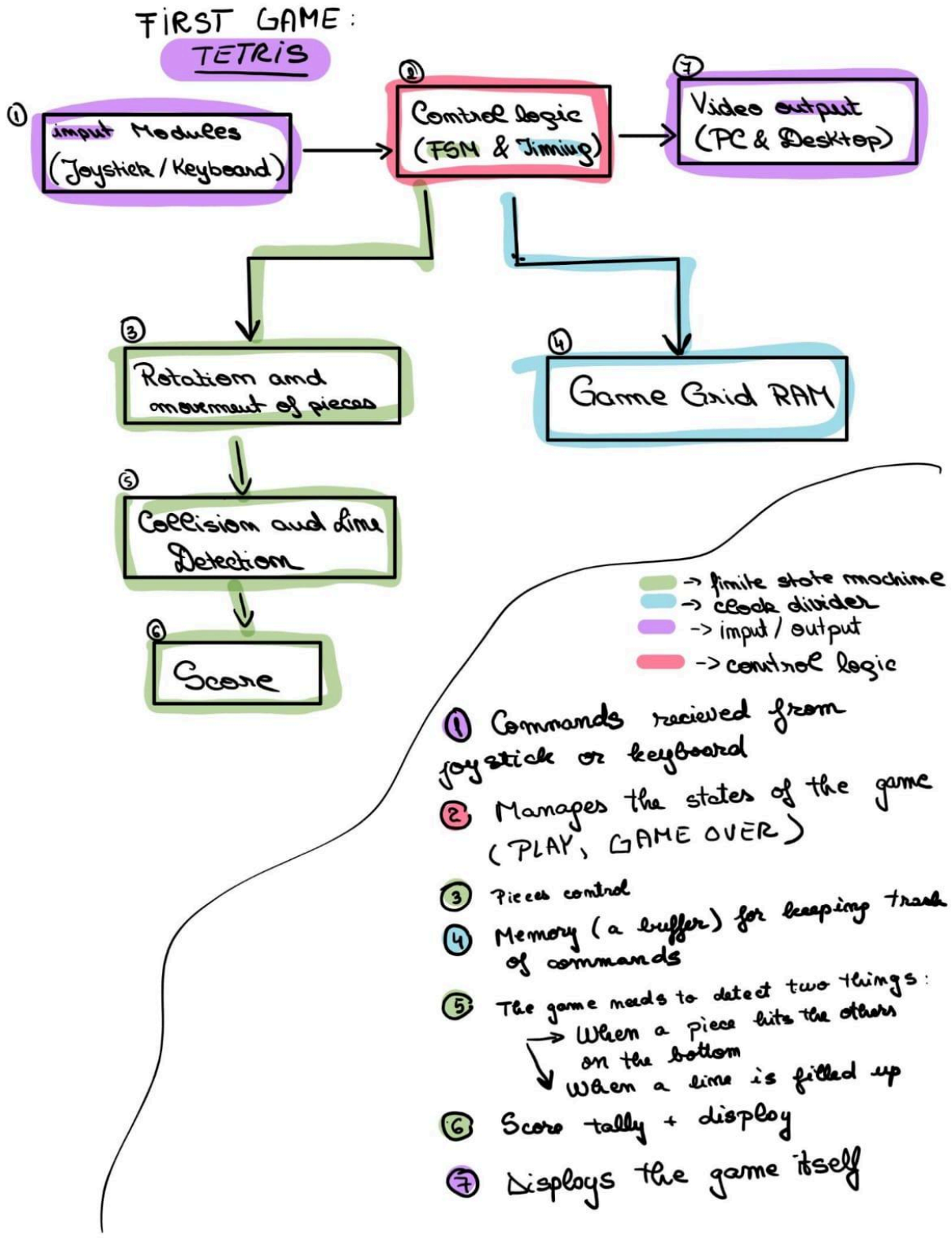
❖ **Implementation Overview**

To implement these games, we will:

- Use **Verilog HDL** to program the FPGA.
- Design separate **FSMs** for each game to handle states like start, play, win, and game over.
- Develop a **VGA synchronization module** to output frames in real time.
- Manage game data (like tile states or falling piece positions) using **registers and arrays** in hardware.
- Interface the keyboard via **PS/2 protocol**, decoding input in real-time.

Each game will be modular and loadable from a top-level controller based on menu selection. The entire system will be structured for **reusability and scalability**, allowing future games or features to be added with minimal architectural changes.

2) Block Diagrams



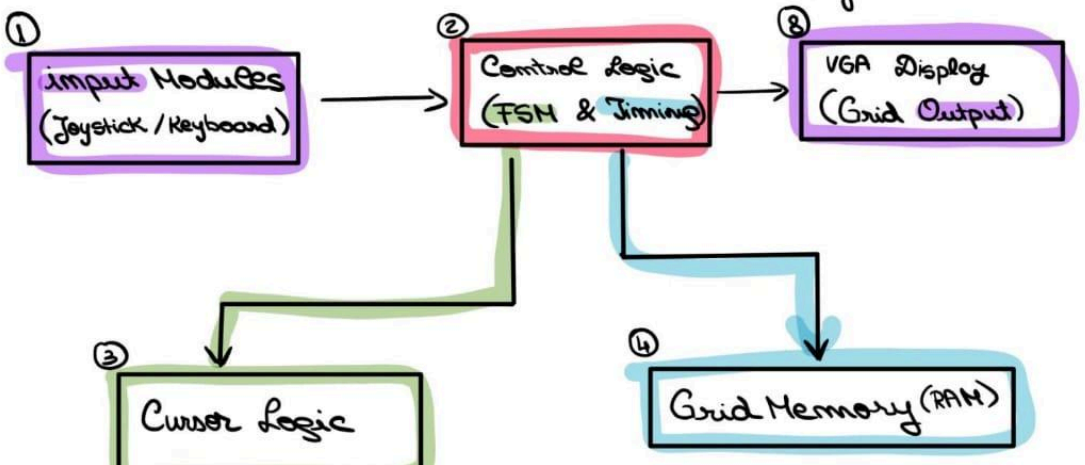
SECOND GAME:

MINESWEEPER

OUR VERSION:

LOTUS SWEEPER

(where the mines in the original one are lotuses)



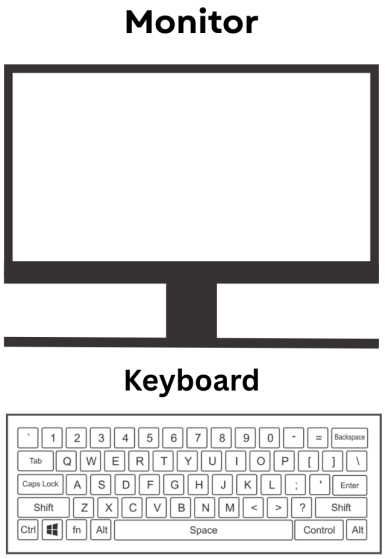
- finit state machine
- clock divider
- control logic
- input / output

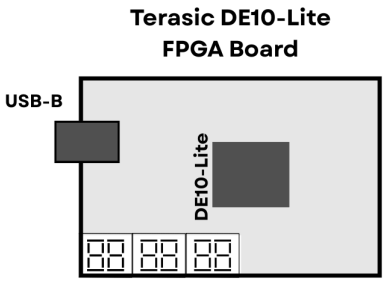
1. Reads inputs from the user
2. Transitioning from game states (WIN / LOSE / PLAYING)
3. Tracks position of cursor on grid
4. Stores the board
5. Random generation of bomb positions
6. The user selects a cell.
   What happens?
   - → it's empty: the neighbouring cells get revealed
   - → it's a bomb: GAME OVER
   - → it's flagged → update cell state
7. Game state FSM
   - → is the game lost or won
   - → resets & transitions
8. Display on monitor (output)

3) Project specifications

COMPONENTS:

- Terasic DE10 Lite FPGA Board
- Monitor
- Keyboard





INPUT: Keyboard
The arrows will be used to move the tetris pieces (Tetrominoes) and for navigating the Lotus Sweeper (Minesweeper) map.
The F key will be used for Leaves (Flags) and the L key will be used for Lotus view mode (debug/extended use).

OUTPUT: Monitor
The FPGA will display the games states on a monitor, the falling Tetromino will be stored within an FSM which will encode information about its center coordinate, shape, and rotational orientation. Every position refresh, the state will update based on the player's inputs.

4) **Software Models**

a) **Tetris:**

1)FSM:

1.START(with GENERATE/SPAWN): waiting for the start button to be pushed and generate a new block
2.DROP: current block drops down the board
3.LOCK: the block is locked on it's position
4.CHECK_LINE:evaluate if a line is complete
- **CLEAR_LINE(with SHIFT_DOWN):**if a line is full, clear it. Shift everything above it down.

5.CHECK_HEIGHT:check if the height of the blocks reaches the top.
- **DROP:** if not, continue dropping the next block. (~back to 2)
- **GAME_OVER:**if the height limit is reached. Playes loses.

2) Data structures:

- ***Game board***: A 2D grid representing the playfield.
- ***Tetromino Pieces***: A set of 7 shapes, each with 4 possible rotation states.
- ***Current Piece***: Tracks the active piece's type, rotation state and position on the board.
- ***Next Piece Queue (optional)***: Stores upcoming pieces for preview or planning.

3) Game Logic Functions:

- **Movement Checking**: Verifies if the active piece can move left, right, down, or rotate based on the board state.
- **Collision Detection**: Determines whether the active piece collides with the walls, bottom, or other blocks.
- **Locking Mechanism**: Integrates a landed piece into the board when it can no longer fall.
- **Line Clearing**: Detects full horizontal lines and removes them. Shifts remaining rows down.
- **Game Over Condition**: Triggered if a new piece overlaps the top of the board.

b) Lotus Sweeper:

1) FSM:

1.START: begin the game
2.GENERATE: create board with hidden cells and lotus
3.INPUT: player selects a cell
4.REVEAL: reveal what is in the selected cell
5.CHECK_CELL:
- **GAME_OVER:** player loses if a lotus is revealed
- **UPDATE:** update the board with revealed safe cells

6.CHECK_WIN: see if all non-lotus cells are revealed
- **COMPLETE:** player wins if all non-lotus are revealed
- **INPUT:** if not all non-lotus are revealed

*Blue states are conditional checks (if-statements) that must be passed in order for the game to proceed.

2) Data structures:

- ***Game board***: A 2D grid representing the playfield.
- ***Cells:*** each one stores:
  1. Whether it contains a lotus. (boolean)
  2. Whether it has been revealed. (boolean)
  3. Whether it is flagged. (boolean)
  4. Number of adjacent mines. (numbers of neighboring mines)

3) Game Logic Functions

1. Initialize Board

- Sets up a new game.
- Randomly places a fixed number of mines on the grid.
- Calculates how many mines are adjacent to each cell.

2. Reveal Cell

- Uncovers a cell chosen by the player.
- If it's a mine: the game ends (loss).
- If it has zero adjacent mines: automatically reveals surrounding cells (cascading reveal).
- If it has a number: just shows that number.

3. Toggle Flag

- Adds or removes a flag on a specific cell.
- Used by the player to mark suspected mines.
- Only allowed on unrevealed cells.

4. Check Win Condition

- Determines if the player has won.
- Win occurs when all non-mine cells are revealed.
- (Optional rule: all mines are flagged correctly.)

5. Game Over

- Triggered when a mine is revealed.
- Stops all interaction.
- Optionally reveals all mines for the player.

6. Reset Game

- Clears the current board and state.
- Prepares for a new game with fresh mine placement.

5) Work breakdown

To maximize efficiency, we decided to split the team into three main departments:

- Logic (control logic): Flavia & Monica
- Coding (implementation): Ingrid & Bianca
- Design (visual output): Andreea

- **The logic department** (green and blue tracks from the block diagrams) maps the overall game flow using boolean algebra and logic gates.

- **The coding department** implements the control logic in Verilog.

- **The design department** works on the visual output using pixel art.

6) Timeline

2 weeks (week 10):

Architecture, game mechanics and simulation.

Game/ Department	Logic	Code	Design
Tetris	Design finite state machines for piece control and grid handling.	Write test benches for logic FSMs. Set up simulation for VGA signal generator module	Mock-up screen layout and piece design.
Lotus Sweeper	Design finite state machines for lotus placements and cell states.		

4 weeks (week 12):

Prototyping and hardware integration.

Game/Department	Logic	Code	Design
Tetris	Simulate grid logic and FSMs.	Run module simulations. Display integration (map logic data to VGA).	Polished game display.
Lotus Sweeper			

6 weeks (week 14):

Final tests and demo.

Game/Department	Logic	Code	Design
Tetris	Final bug fixing.	Final game structure ready.	Final UI/UX designs.
Lotus Sweeper			

Github repository link:
https://github.com/bianca-popescu/game-arcade-FPGA