Space Invaders Game

Cornelius Gruss (cgruss), Ayla Tabi (aylatabi)
https://github.com/aylatabi/lab5

Abstract
This project presents a Space Invaders-inspired arcade game controlled with Xbox controllers interfaced with the BeagleBone Black. This game supports single and multiplayer modes where players collaboratively defend against aliens that return fire with laser beams. Players control platforms to eliminate all of the aliens while protecting their health with shields. Victory is achieved by destroying all aliens before the player's health is depleted. The accomplishments include successfully integrating Xbox controller input with the BeagleBone black through custom kernel module implementation and userspace drivers. Additionally, we developed a fully multithreaded game using atomic variables and locks for thread safe synchronization, successfully preventing race conditions across all threads.

1. Introduction

Project Topic

This project implements a classic multiplayer Space Invaders arcade game on the BeagleBone. The game features real-time graphics rendered using the Qt framework, hardware input from Xbox controllers via the Linux input subsystem, and a multi-threaded architecture managing concurrent game events including player movement, projectile physics, and alien attacks.

Motivation

The motivation for this project was to bring people together through an in-person multiplayer gaming experience. The original Space Invaders arcade game was single-player only, so we wanted to create a two-player cooperative version where players can team up against the alien invasion. By using Xbox controllers—familiar hardware that most people have used—we aimed to make the game accessible to players of all ages through intuitive gameplay.

Space Invaders was an ideal choice because it's a recognizable retro classic that's simple enough to pick up immediately but still engaging. Implementing it on an embedded platform also provided practical experience with real embedded systems techniques: configuring the Linux kernel to enable the xpad driver module (via menuconfig), handling real-time hardware input through the Linux input subsystem, and managing multiple concurrent threads for smooth 60 FPS gameplay.

Big Picture Overview

The system consists of three main components:

1. **Input Layer**: Xbox controllers connected via USB, using the Linux xpad kernel driver and accessed through /dev/input/event* device files with Qt's QSocketNotifier for event-driven input handling
2. **Game Logic Layer**: Multi-threaded C++ implementation with separate threads for display updates (60 FPS), player platform movement, cannon projectiles (one per player), and four concurrent alien attack threads
3. **Rendering Layer**: Qt-based 2D graphics using QPainter, drawing pixel-art style sprites for aliens, player platforms, projectiles, shields, and explosion animations

Results Summary

We successfully implemented a fully functional two-player Space Invaders game running at 60 FPS on the BeagleBone Black. The implementation manages 8+ concurrent threads with proper mutex synchronization, handles real-time Xbox controller input with sub-frame latency, and provides smooth gameplay with collision detection, shield mechanics, and animated explosions. The game supports both single-player and two-player modes with independent controller input.

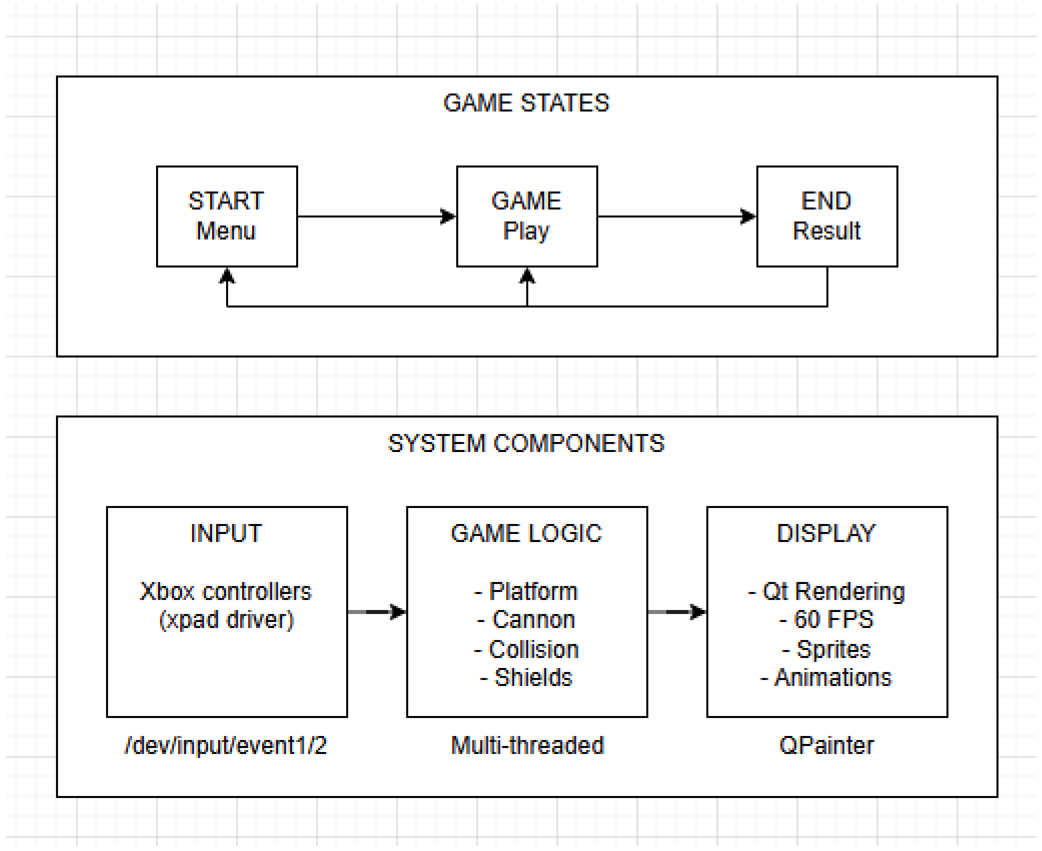
2. Design Flow

Project Components

The project consists of the following components (detailed implementation in Section 3):

1. **Xbox Controller Integration**: Kernel configuration and input handling
2. **Platform/Cannon System**: Player movement and shooting mechanics
3. **Alien System and Attack Threads**: Enemy behavior and concurrent attack logic
4. **Collision Detection**: Hit detection between projectiles and game objects
5. **Shield System**: Defensive player ability with cooldown
6. **Explosion Animation**: Visual effects for destroyed players
7. **Start and End Display Screens**: Game state UI and menu system
8. **Multi-threading Design**: Thread architecture and synchronization

Flow Chart



Team Contribution

Component	**Implemented By**
Xbox Controller Setup (xpad/menuconfig)	Ayla and Cornelius
Platform Movement & Cannon Firing	Ayla
Multi-threading Architecture	Ayla
Alien Attack Threads	Cornelius
Collision Detection (Hitting Aliens)	Cornelius
Shield System	Cornelius
Start/End Display Screens	Ayla

Overall Contribution: Ayla Tabi 50% and Cornelius Gruss 50%

3. Project Details

1. **Xbox Controller Integration** (See spaceinvaders.cpp: onControllerInput_player1(), onControllerInput_player2())

   The Xbox controllers connect to the BeagleBone via USB and are managed by the Linux xpad kernel driver. To enable controller support, we downloaded the stock BeagleBone image and used menuconfig to enable the xpad driver module in the kernel configuration.

   The game accesses controllers through the Linux input subsystem at /dev/input/event1 (Player 1) and /dev/input/event2 (Player 2). Controllers are opened with non-blocking I/O flags (O_RDONLY | O_NONBLOCK) to prevent the game from freezing while waiting for input.

   Qts QSocketNotifier monitors the file descriptors and triggers callback functions (onControllerInput_player1(), onControllerInput_player2()) when input events are available. Each event is read as a struct input_event from <linux/input.h>, containing the event type, code, and value.

2. **Platform/Cannon System** (See spaceinvaders.cpp: platformThread_func(), player1_cannonThread_func(), player.cpp)

   The Player class manages each player's position, health (max 30), and movement boundaries (0-440 pixels). A dedicated platform thread continuously reads the analog stick value and updates player position at 4 pixels per frame when the stick exceeds a threshold of ±8000.

   Cannon firing is handled by separate threads for each player (player1_cannonThread, player2_cannonThread). When the A button is pressed, the cannon projectile travels upward in 20-pixel increments with 48ms delays per frame. The projectile is drawn as a 4×5 pixel rectangle.

   Thread safety is maintained using a platform_mtx mutex to protect position and health updates from race conditions.

3. **Alien System and Attack Threads** (See spaceinvaders.cpp: attackThread_func(), alien.h)

   Aliens are organized in a 3-row grid with 8, 7, and 6 aliens per row respectively. Three alien types exist with different sizes and colors:
   - Type 0: 48×32 pixels, yellow
   - Type 1: 32×32 pixels, magenta

- Type 2: 44×32 pixels, cyan

Four concurrent attack threads (attackThread[0-3]) fire projectiles at the player. Each thread:

1. Waits 2 seconds before starting attacks
2. Selects a random living alien
3. Fires a projectile downward in 10-pixel increments
4. Waits a random delay (400-1100ms) before the next attack

The alien_mtx mutex protects the alien state during selection and damage application.

4. **Collision Detection** (See spaceinvaders.cpp: hit_alien_row_1(), hit_alien_row_2(), hit_alien_row_3())

   Three collision detection functions handle hits for each alien row:
   - hit_alien_row_1(): Top row (8 aliens)
   - hit_alien_row_2(): Middle row (7 aliens)
   - hit_alien_row_3(); Bottom row (6 aliens)

   When a cannon projectile reaches an alien row's Y-coordinate range, the function checks if the projectile's X-position falls within any aliens column bounds. On collision, the alien takes 100 damage and is disabled when health reaches zero.

   Player collision detection occurs when alien projectiles reach Y ≥ 235 (bottom of screen). The game checks if the projectile X-position overlaps with the player's 40-pixel-wide hitbox.

5. **Shield System** (See spaceinvaders.cpp: paintEvent(), shield logic in game loop)

   Players can activate a shield by pressing the X button, providing 3 seconds (180 frames) of damage immunity. The shield is displayed as a white horizontal bar above the platform.

   After the shield expires, a 5-second (300 frames) cooldown period begins. During cooldown, yellow indicator blocks appear progressively to show remaining cooldown time:
   - Frames 0-100: 1 block
   - Frames 100-200: 2 blocks
   - Frames 200-300: 3 blocks (ready)

6. **Explosion Animation** (See spaceinvaders.cpp: drawExplosion())

   When a player's health reaches zero, a three-phase explosion animation plays over 60 frames:

- Phase 1 (frames 1-15): Core collapse – 4 pixels at center
- Phase 2 (frames 16-35): Burst outward – 6 particles expanding from center
- Phase 3 (frames 36-60): Full explosion – 5 particles scattered across screen

After the animation completes, persistent wreckage blocks remain at the destruction point. The explosion color matches the player (pink for P1, green for P2).

7. **Start and End Display Screens** (See spaceinvaders.cpp: paintEvent(), GameState enum)

   The game uses a state machine with three states: START, GAME, and END.

   Start Screen: Displays the game title and prompts players to select single-player (1P) or two-player (2P) mode using the A button.

   End Screen: Shows "GAME OVER" when all players are defeated or "YOU WIN" when all aliens are destroyed. Players can press A to replay or return to the start menu.

8. **Multi-threading Design** (See spaceinvaders.cpp: startGameThreads(), stopGameThreads())

   The game uses 8+ concurrent threads:

Thread	Purpose	Update Rate
Display Thread	Trigger Qt repaints	16ms (60 FPS)
Platform Thread	Player Movement	16ms
Player 1 Cannon	P1 projectile	On button press
Player 2 Cannon	P2 projectile	On button press
Attack Threads (x4)	Alien Projectiles	400 - 1100ms random

   Thread synchronization uses:
   - **Mutexes** (platform_mtx, alien_mtx) for shared game state
   - **Atomic variables** for controller input values (lock-free communication)
   - **Thread joining** for cleanup when game ends

Figures below show a technical flow chart for each state: START, GAME, END. These charts show how the threads and controller inputs interact with each other as well as other functions/classes in our code.

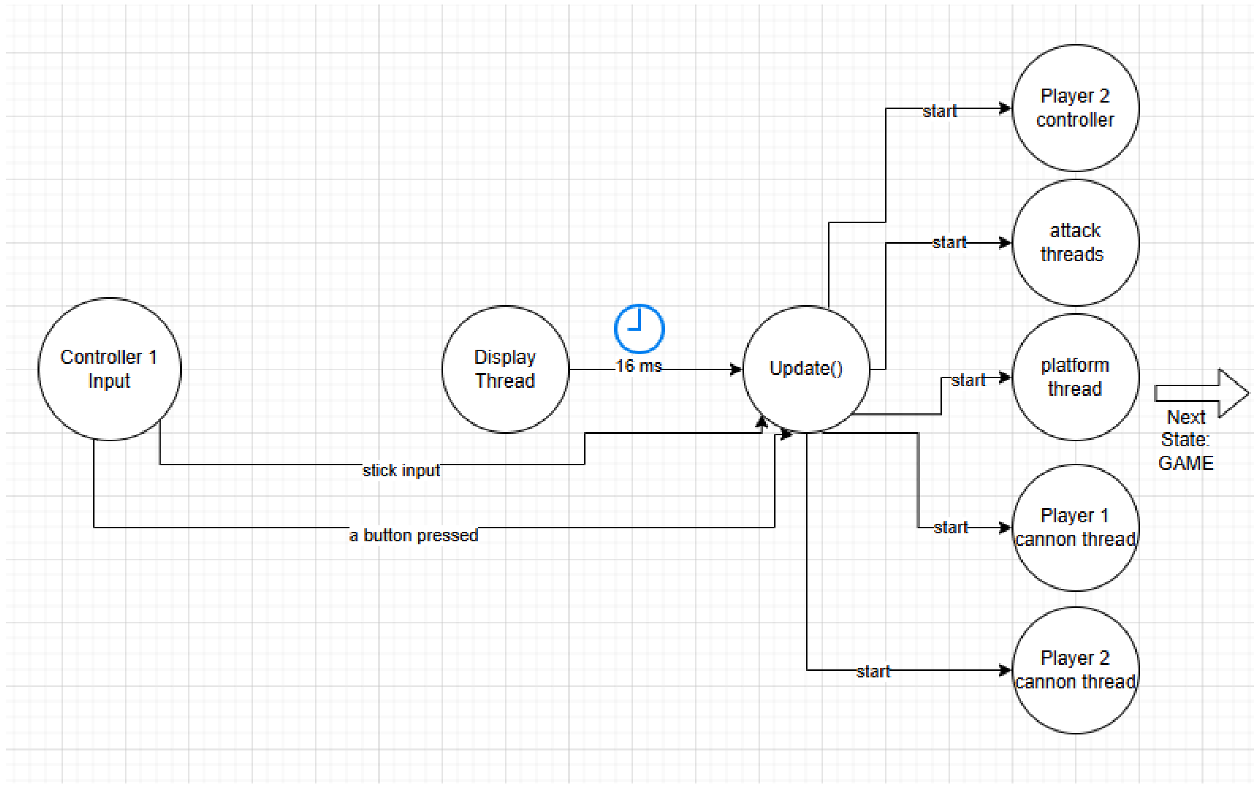


Figure 1: Flowchart for the START state

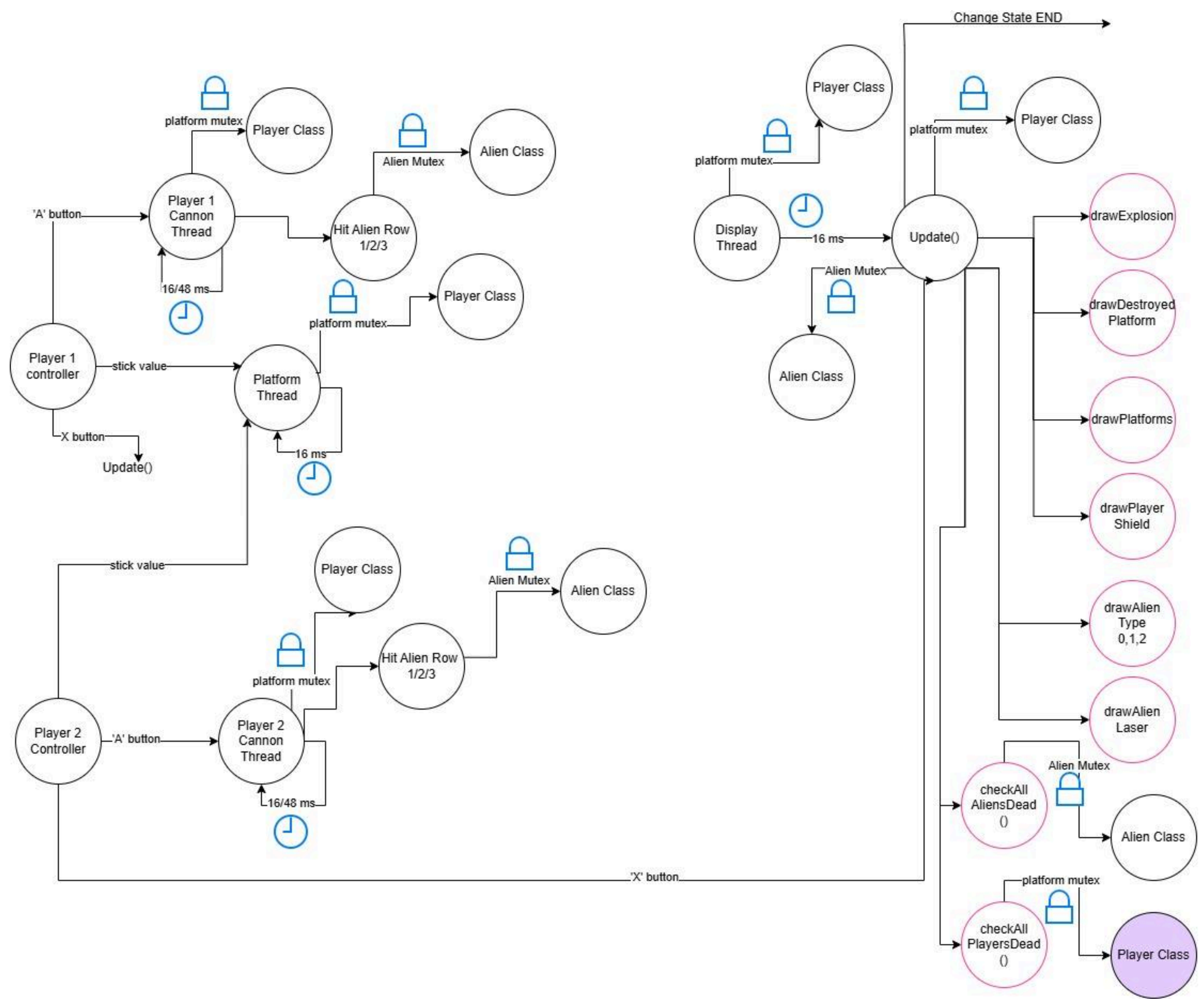


Figure 2: Flowchart for GAME state

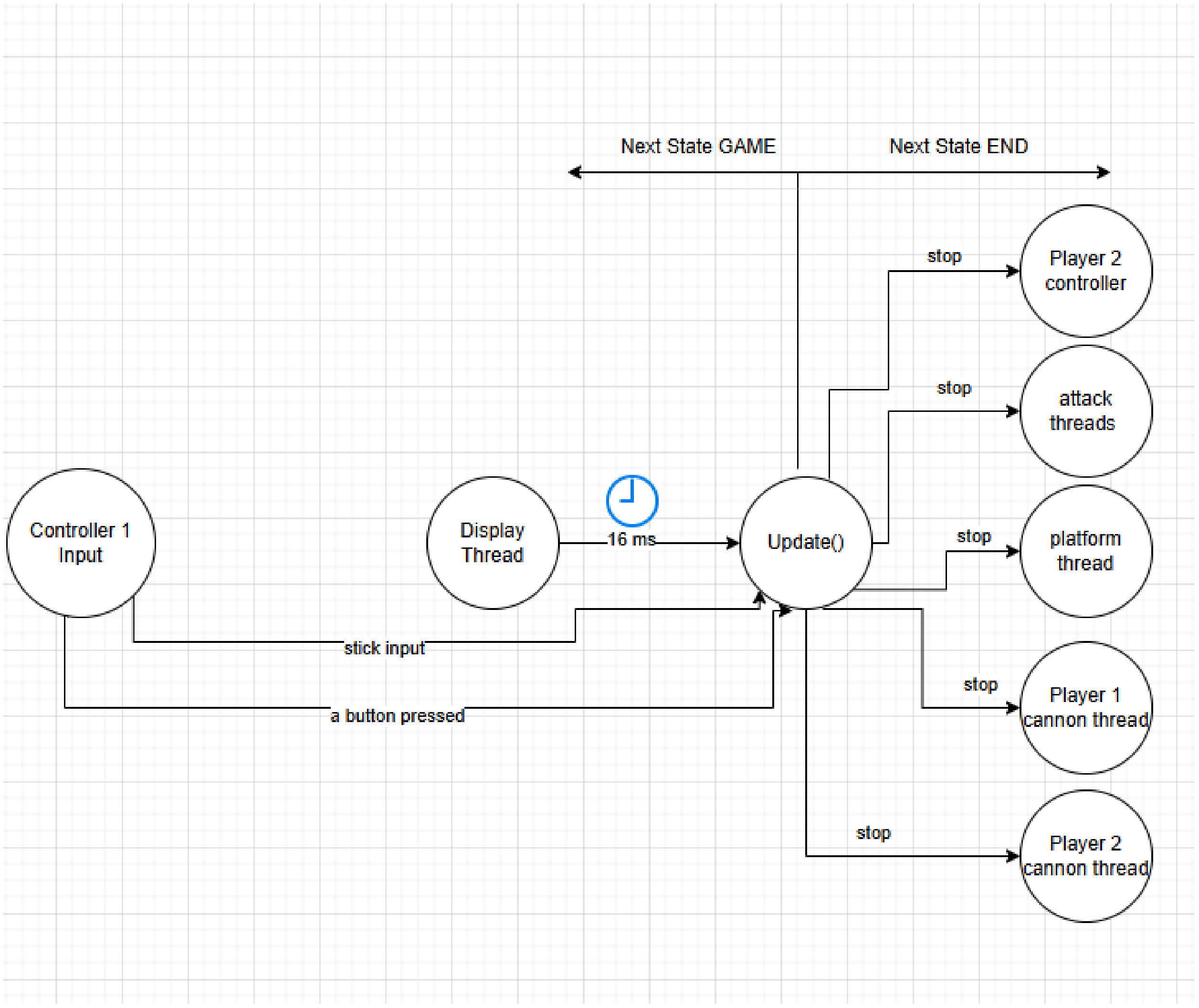


Figure 3: flow chart for END state

4. Summary

We successfully implemented a two-player Space Invaders game on the BeagleBone Black embedded platform. Key accomplishments include:

- Configured the Linux kernel to enable Xbox controller support via the xpad driver module
- Implemented real-time input handling using Qt's QSocketNotifier and the Linux input subsystem
- Designed a multi-threaded architecture with 8+ concurrent threads running at 60 FPS
- Developed collision detection, shield mechanics, and explosion animations
- Created a complete game experience with start menu, gameplay, and end screens

Potential improvements for future development:

- **Alien movement:** Currently aliens are stationary; adding horizontal/descending movement would increase difficulty
- **Score system:** Tracking and displaying player scores
- **Dynamic difficulty:** Adjusting attack frequency based on remaining aliens or users strength

References

[1] Qt Project. "Qt Documentation." https://doc.qt.io/
[2] P.Oj. "xpad - Xbox Controller Driver for Linux." https://github.com/paroj/xpad
[3] cppreference.com. "C++ Thread Support Library." https://en.cppreference.com/w/cpp/thread