ASSEMBLY LANGUAGE GAME: BLOCKBUSTER

An *Arcade Game* presented to the

College of Information System and Technology Management

Pamantasan ng Lungsod ng Maynila

In Partial Fulfillment of the Requirements for

CSC 0222-2 Computer Architecture and Assembly Language Programming

Prepared by: BSCS 2nd Year Block 1 — Group 2

Abundo, Jonalene Ryza B. 2023-34050	Mabutas, Carla R. 2023-34029
Apelledo, Mark Daniel D. 2022-59004	Gernale, Xienen Lei M. 2023-34092
Dalangin, Hershey Anne S. 2023-34042	Vaflor, Mariel Kim D. 2022-34057
Camus, Angel Lyn M. 2022-59010	Miranda, Mariebethel Roussamiere C. 2022-59032

Presented to: **Engr. Elsa S. Pascual**

May 17, 2024

Table of Contents

Members' and Assigned Tasks

Members	Roles
Abundo, Jonalene	o Documentation Lead o Focused on delegating tasks for documentation o Proposed ideas for power-ups
Apelledo, Mark Daniel	o Oversaw project timeline and team coordination o Full-Stack Developer o Delegated tasks and contributed to UI and game logic
Dalangin, Hershey Anne	o Created the brochure o Front-End Developer o Suggested nerfs for the game
Camus, Angel Lyn	o Full-Stack Developer o Integrated power-up and nerf systems in both gameplay modes
Miranda, Mariebethel R.	o Lead Front-End Developer o Utilized Figma and redesigned the main page of the game
Mabuta, Carla	o Front-End Developer o Created the presentation (PPT) o Suggested nerfs for the game
Gernale, Xienen Lei	o Created visual animations for power-ups and level transitions o Assisted the Lead Front-End Developer in modifying the main page
Vaflor, Mariel Kim	o Front-End Developer o Created the presentation (PPT) o Suggested power-ups for the game

I. Introduction

The video game industry has undergone a remarkable transformation since its inception, evolving from simple pixel-based experiences to complex, immersive digital worlds. This evolution is particularly evident in the arcade genre, where classic games continue to inspire modern developers while maintaining their timeless appeal. Among these influential titles is *Breakout* (1976), developed by Atari, which revolutionized gaming by introducing the brick-breaker genre. Its simple yet engaging mechanics captured the imagination of players worldwide and laid the foundation for countless similar games.

Drawing inspiration from this iconic legacy, **BlockBuster** emerges as a modern interpretation of the *Breakout* formula. Developed in x86 Assembly Language as a comprehensive course project for *Computer Architecture and Assembly Language Programming*, the game serves as both an educational tool and a testament to what can be accomplished using low-level programming. This challenging yet rewarding approach emphasizes core principles of computer architecture.

The original version of **BlockBuster**, despite being constrained by the limits of assembly language and hardware, implemented features such as:

- A progressive difficulty system across multiple levels
- Different gameplay modes
- Functional audio-visual feedback

These demonstrated technical proficiency and creativity in the face of limitations.

However, the first iteration faced several issues:

- Restrictive four-character player name input
- Static difficulty progression
- A basic user interface

These challenges, while understandable due to assembly language limitations, offered opportunities for enhancement.

II. Enhancements

The enhanced version of **BlockBuster** introduces several key improvements that elevate the gaming experience while preserving the engaging mechanics of the original. These enhancements address prior limitations and introduce new gameplay elements that increase depth, strategy, and replayability.

1. Expanded Name Capacity System *(+1 Added Enhancement)*

The original four-character player name limitation has been significantly upgraded to allow for better personalization and usability.

- **Extended Character Limit**
  Increased from **4** to **20 characters**, enabling players to use full names or gaming aliases.
- **Enhanced Personalization**
  Promotes stronger player-game connection through more identifiable names.

2. Dynamic Power-Up System *(+1 Added Enhancements)*

A new system introduces power-ups that provide strategic advantages and gameplay variety.

Power-Up Types:

- **Longer Paddle Modification**
  - Increases paddle size
  - Better control

3. Challenge Modifiers *(+2 Added Enhancements)*

To balance power-ups and maintain gameplay difficulty, challenge modifiers (nerfs) were introduced.

Modifier Types (Nerfs):

- **Shortened Paddle Modification**
    - Decreases paddle size
    - Demands precise control
- **Time Penalty for Missed Ball**
    - Subtracts 5 seconds from the timer each time the ball falls
    - Increases time pressure and urgency
    - Encourages careful paddle control and precision

4. Visual Improvements

The visual presentation of the game was enhanced by incorporating more vibrant and colorful elements, while still preserving the classic arcade aesthetic.

- **Main Menu Redesign** *(+1 Added Enhancement)*
    - Maintains retro theme with updated polish
- **P button Keyboard** *(+1 Added Enhancement)*
    - It supports a **pause/play system** and features a **pause menu** triggered by pressing ***'P'***

III. Definition of Terms

1. **BlockBuster -** a modern reimagining of the classic *Breakout* arcade game, developed using x86 Assembly Language. This enhanced version features multiple gameplay modes, progressive difficulty levels, and various power-up systems, all implemented within the constraints of low-level programming.
2. **Breakout** – the original arcade game developed by Atari in 1976, which established the brick-breaker genre. It introduced the core mechanic of using a paddle to bounce a ball and break bricks, forming the foundation for countless similar games.
3. **TASM (Turbo Assembler)** – a professional-grade assembler developed by Borland, used for compiling x86 Assembly Language code into executable programs. It includes features like macro support, conditional assembly, and optimization options essential for low-level development.
4. **DOSBox** – a DOS emulator that enables modern systems to run legacy DOS applications. It is used to run *BlockBuster* by emulating the hardware and operating system environment necessary for the game's execution.
5. **x86 Assembly Language** – a low-level programming language that allows direct interaction with hardware. In *BlockBuster*, it is used to implement game mechanics, manage memory, process inputs, and demonstrate fundamental computer architecture concepts.
6. **Power-ups** – temporary gameplay enhancements that give players strategic advantages. In *BlockBuster*, these include the Bigger Ball, Slower Motion, and Time Freeze, each offering unique benefits while preserving gameplay balance.
7. **Nerfs** – gameplay modifiers that increase difficulty to balance the presence of power-ups. These include Reduced Hearts, Faster Motion, and Shortened Paddle, encouraging players to improve their skills and adapt to challenges.
8. **Progressive Difficulty** – a game design principle where challenge increases as the player progresses. In *BlockBuster*, this is seen through faster ball speeds, complex brick arrangements, and layered modifiers, ensuring long-term engagement.

IV. How to Play

The game begins with a landing panel where the player can choose between starting the game by selecting the **Play** button or viewing the **Controls** to learn how to play. Before proceeding, the player is prompted to input their name, with a maximum character limit of 20. Upon entering the main menu, the player can choose between two game modes: **Level Mode** and **Timed Mode**. In addition, there is an **Options** section where players can adjust sound settings to their preference.

In **Level Mode**, the speed of the game gradually increases with each level, creating a more challenging gameplay experience. The core mechanic requires the player to use the **arrow keys** to control a paddle and catch a ball, bouncing it upward to break the blocks above. Each round begins with **three lives**, and a life is lost each time the player fails to catch the ball. In **Timed Mode**, the player attempts to break as many blocks as possible before the countdown timer runs out. Throughout gameplay, **power-ups** and **nerfs** may fall from broken blocks, and the player must catch these falling items to either gain advantages or adapt to new challenges. These mechanics aim to enhance both strategic play and being replayable.

Game Modes

1. Level Mode – the traditional progression-based experience, ideal for players who enjoy methodical advancement and skill development.

- **Level Structure**:
  - Five progressively challenging levels
  - Unique brick pattern per level
  - Increasing difficulty through: Faster ball movement and Limited lives
- **Gameplay Mechanics**:
  - Start with a fixed number of lives
  - Complete a level by breaking all bricks
  - Lose a life when the ball falls below the paddle

    - Power-ups and nerfs fall randomly

- **Level Progression**:

  - Level 1: Normal ball speed (good for beginners)
  - Level 2: Slightly increased speed
  - Level 3: Moderate speed — quicker reactions required
  - Level 4: Fast ball movement — more challenge
  - Level 5: Maximum ball speed — ultimate test of skill

2. Timed Mode – a fast-paced, score-driven mode for players who enjoy time-based challenges and quick decisions.

- **Time Management**:
    - Race against the clock

3. Powerplay Mode – a dynamic challenge mode that introduces unpredictability and strategic choices.

- **Gameplay Experience:**
    - Randomly falling power-ups and/or nerfs throughout the match
    - Constant adaptation required based on active modifiers
    - Players must balance risk and reward when collecting items
- **Strategic Challenge:**
    - No guaranteed advantage — nerfs can hinder progress
    - Encourages quick decision-making and situational awareness
    - Adds high replay value through unpredictable modifiers

Control Scheme

The control layout ensures a smooth, accessible gameplay experience.

Primary Controls on Keyboard:

- **Left & Right Arrow Keys** – move the paddle horizontally
  - Allows precise control of ball direction
  - Responsive movement with variable speed
- **Enter** – multi-function key
  - Start the game
  - Confirm selections in menus
- **P button Keyboard** - It supports a **pause/play system** and features a **pause menu** triggered by pressing ***'P'***
- **ESC Key** – game management
  - Return to the main menu
  - Exit the current session

Additional Controls:

- **Sound Toggle**
  - Turn sound effects on/off
  - Adjust volume
  - Enable/disable background music
  - Personalize audio experience

Game Interface

The UI displays essential gameplay information and visual feedback.

Display Elements

- Current score
- Remaining lives
- Time remaining (Timed Mode)
- Active power-ups
- Current level

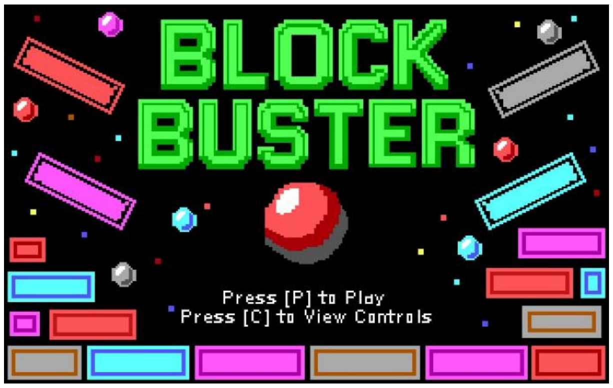
Visual Feedback

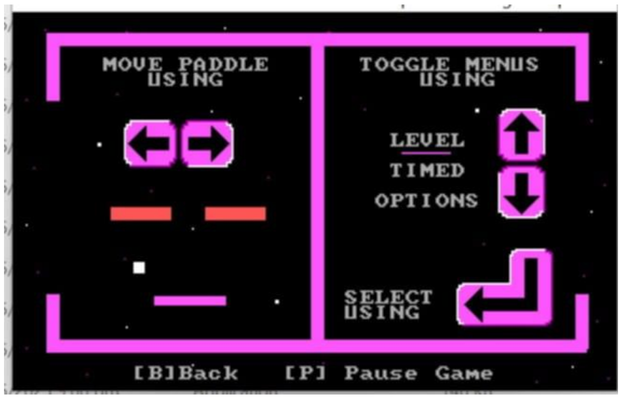
- Ball trajectory indicators
- Power-up status effects
- Life count display
- Score animations
- Level completion indicators
- Pause dialogue box with game options

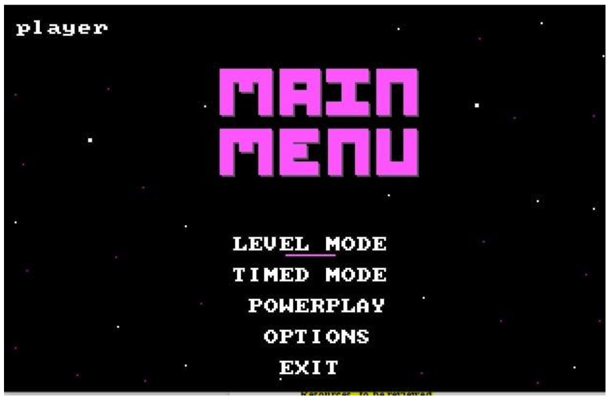
V. Storyboard / UI Design

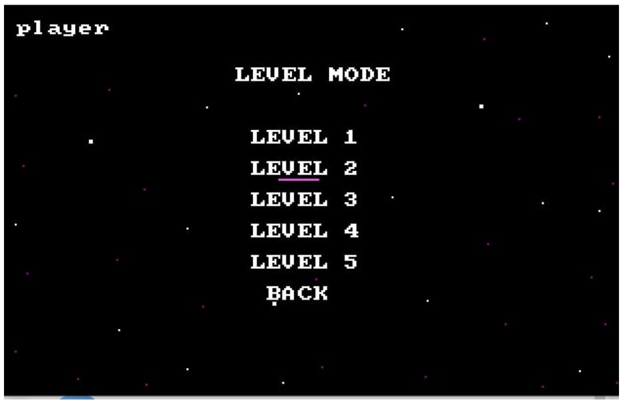
Main Menu

- Game title and logo
- Mode selection
- Name input field
- SFX toggle
- Instructions button









Gameplay Screen

- Brick layout and playfield
- Score and lives display
- Timer (Timed Mode only)
- Power-up and nerf indicators
- Level display

VI. Updated Code in your own GITURL

GitHub Links:

https://github.com/MarkDaniel0702/BlockBusterUpdates.git

https://github.com/MarkDaniel0702/BBuster_NeueVersion.git

VII. Installation Instructions

Required Tools

1. DOSBox Emulator
2. Turbo Assembler (TASM)
3. Notepad++ or equivalent text editor

Steps

1. Install DOSBox: dosbox.com
2. Download TASM Tools
3. Clone the repository:

   git clone https://github.com/jhamped/blockbuster-assembly.git
4. Launch DOSBox and navigate to the game directory
5. Compile using TASM:
   - tasm bbuster.asm
   - tlink bbuster.obj
6. Run the game: bbuster.exe

VIII. References / Credits

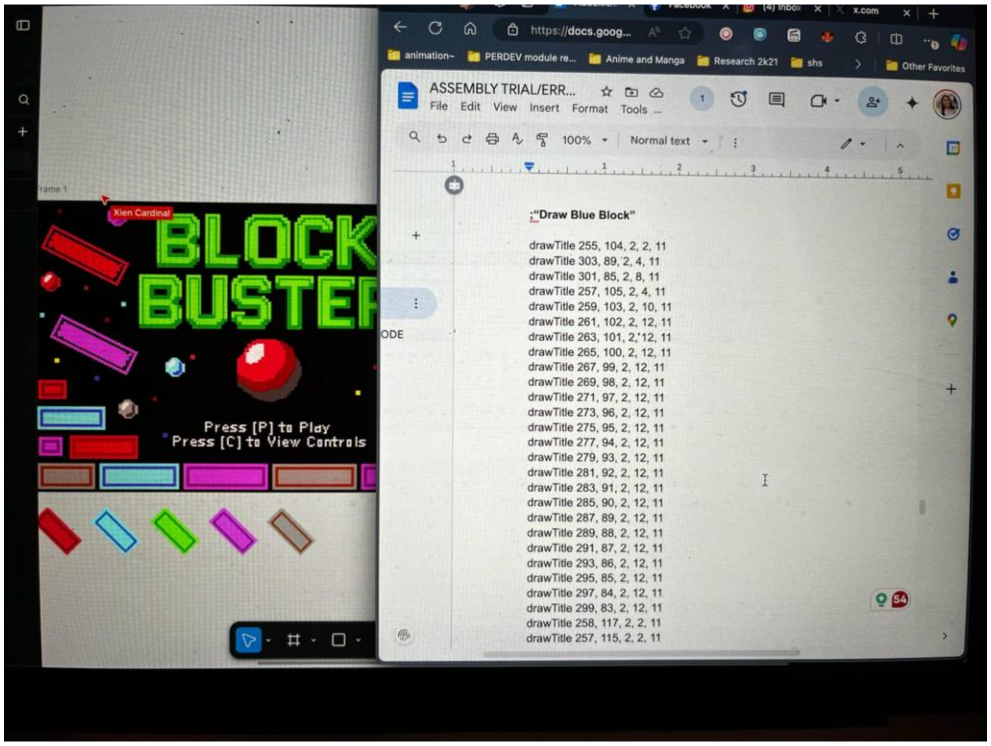
Original Developers – BSCS 2-1 Group 4

- Aguinaldo, Samantha S.
- Bautista, Anna Kathlyn A.
- Chaves, Samantha D.
- De Roxas, Carnation Jhulia I.
- Enriquez, Sybil Mitch S.
- Fallaria, Immaculate L.
- Pantoja, Rhayzel S.
- Pineda, Ernest Michael B.

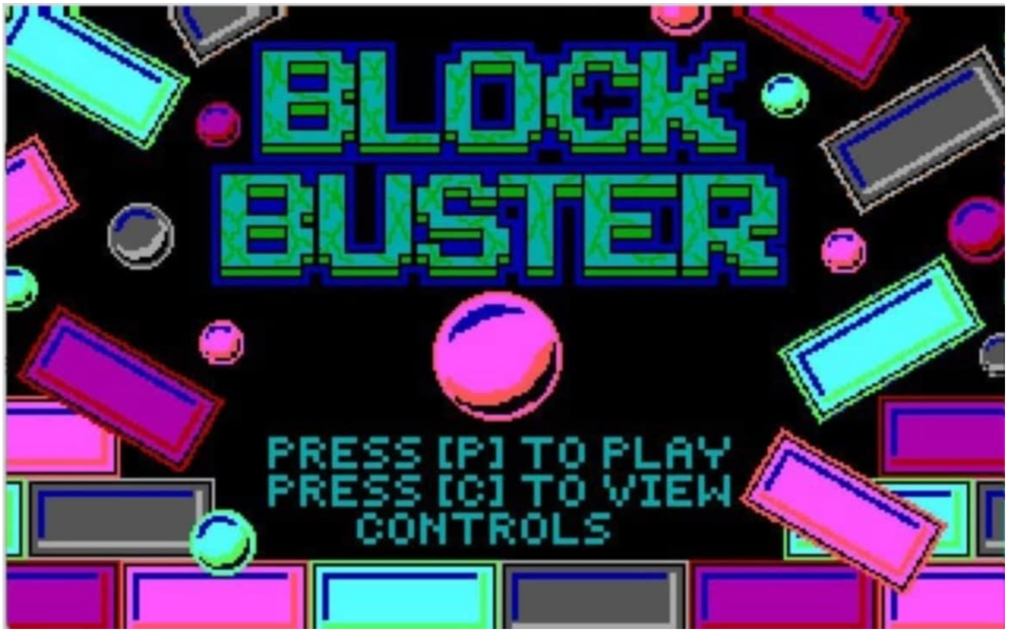
Resources- to be reviewed

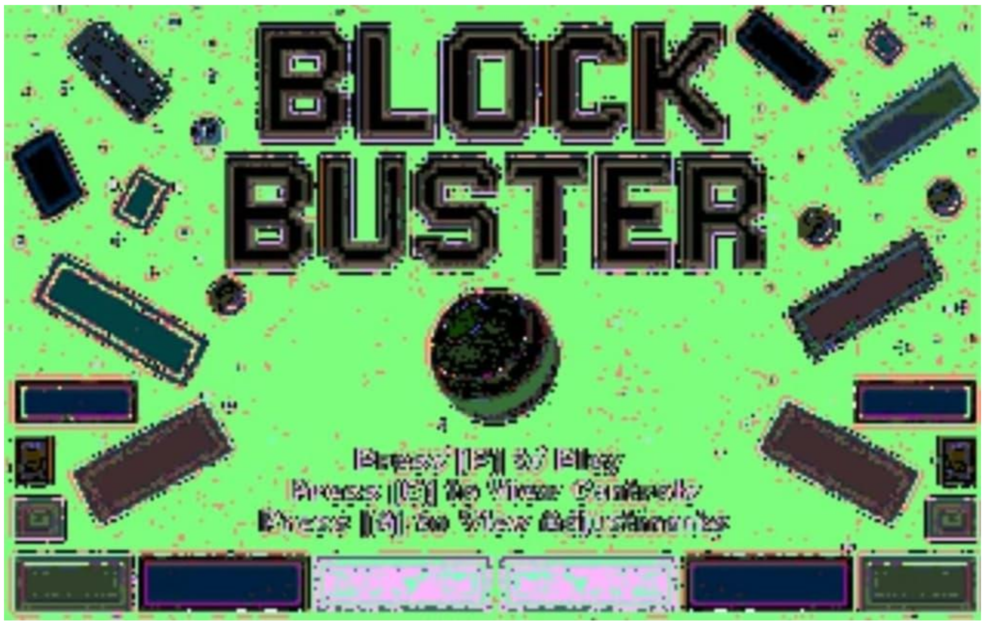
- *Atari Breakout (1976)* – Original game inspiration
- x86 Assembly Language Reference
- DOSBox Documentation
- Turbo Assembler Documentation

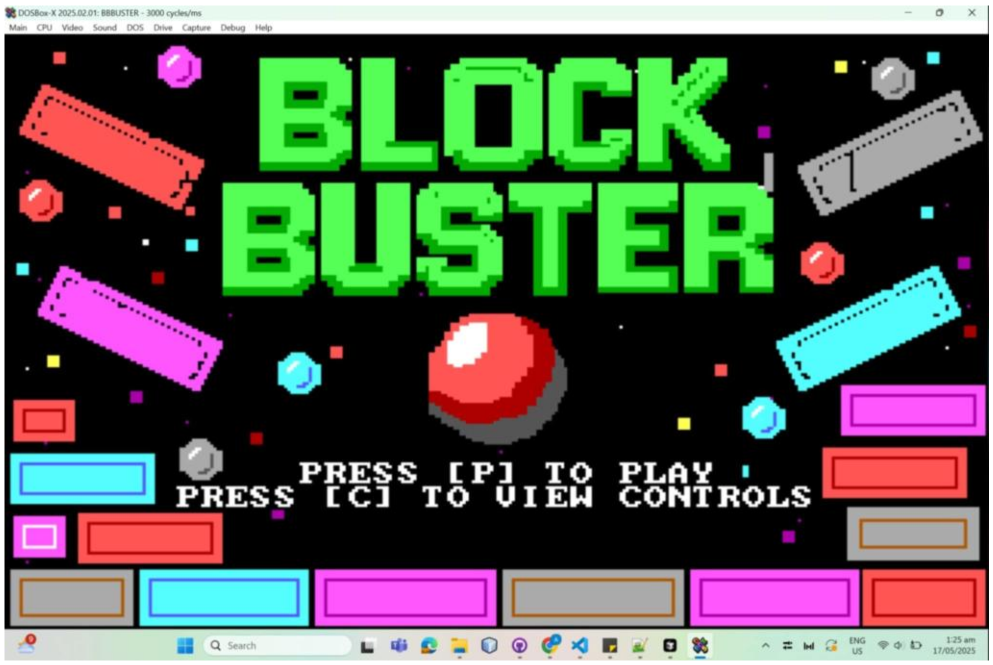
IX. Pictures During Development

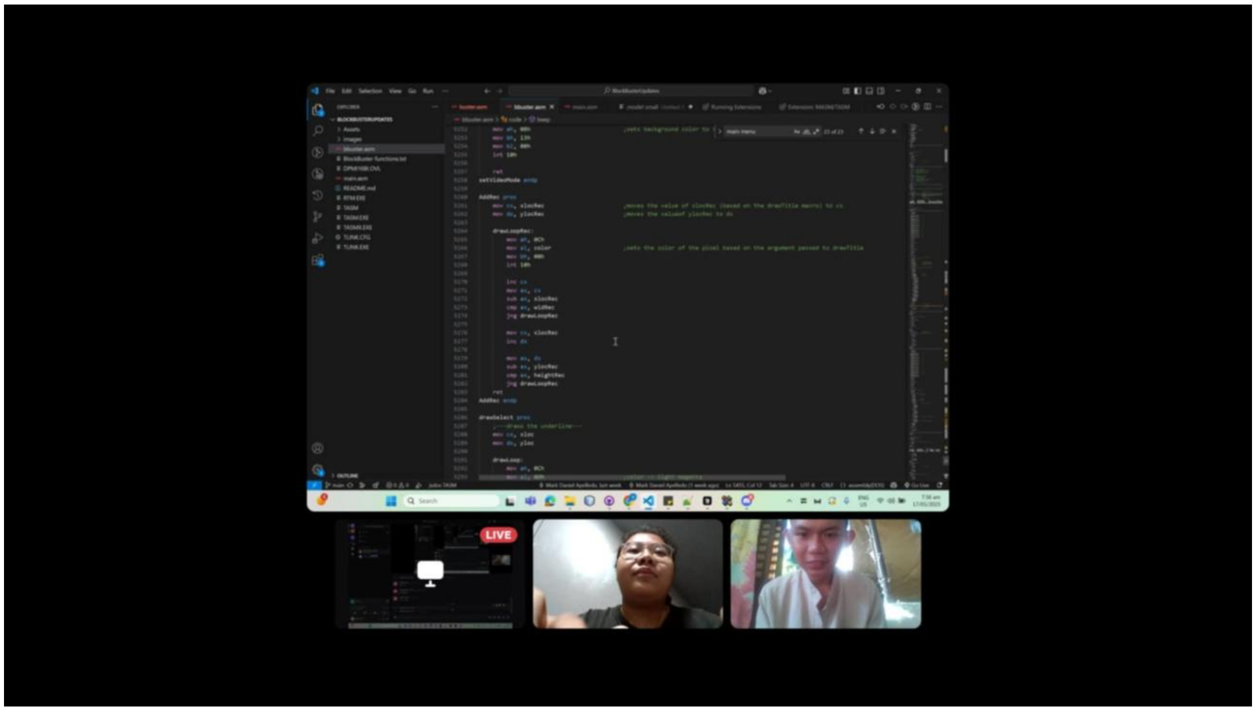


```
;"Draw Blue Block"

drawTitle 255, 104, 2, 2, 11
drawTitle 303, 89, 2, 4, 11
drawTitle 301, 85, 2, 8, 11
drawTitle 257, 105, 2, 4, 11
drawTitle 259, 103, 2, 10, 11
drawTitle 261, 102, 2, 12, 11
drawTitle 263, 101, 2, 12, 11
drawTitle 265, 100, 2, 12, 11
drawTitle 267, 99, 2, 12, 11
drawTitle 269, 98, 2, 12, 11
drawTitle 271, 97, 2, 12, 11
drawTitle 273, 96, 2, 12, 11
drawTitle 275, 95, 2, 12, 11
drawTitle 277, 94, 2, 12, 11
drawTitle 279, 93, 2, 12, 11
drawTitle 281, 92, 2, 12, 11
drawTitle 283, 91, 2, 12, 11
drawTitle 285, 90, 2, 12, 11
drawTitle 287, 89, 2, 12, 11
drawTitle 289, 88, 2, 12, 11
drawTitle 291, 87, 2, 12, 11
drawTitle 293, 86, 2, 12, 11
drawTitle 295, 85, 2, 12, 11
drawTitle 297, 84, 2, 12, 11
drawTitle 299, 83, 2, 12, 11
drawTitle 258, 117, 2, 2, 11
drawTitle 257, 115, 2, 2, 11
```

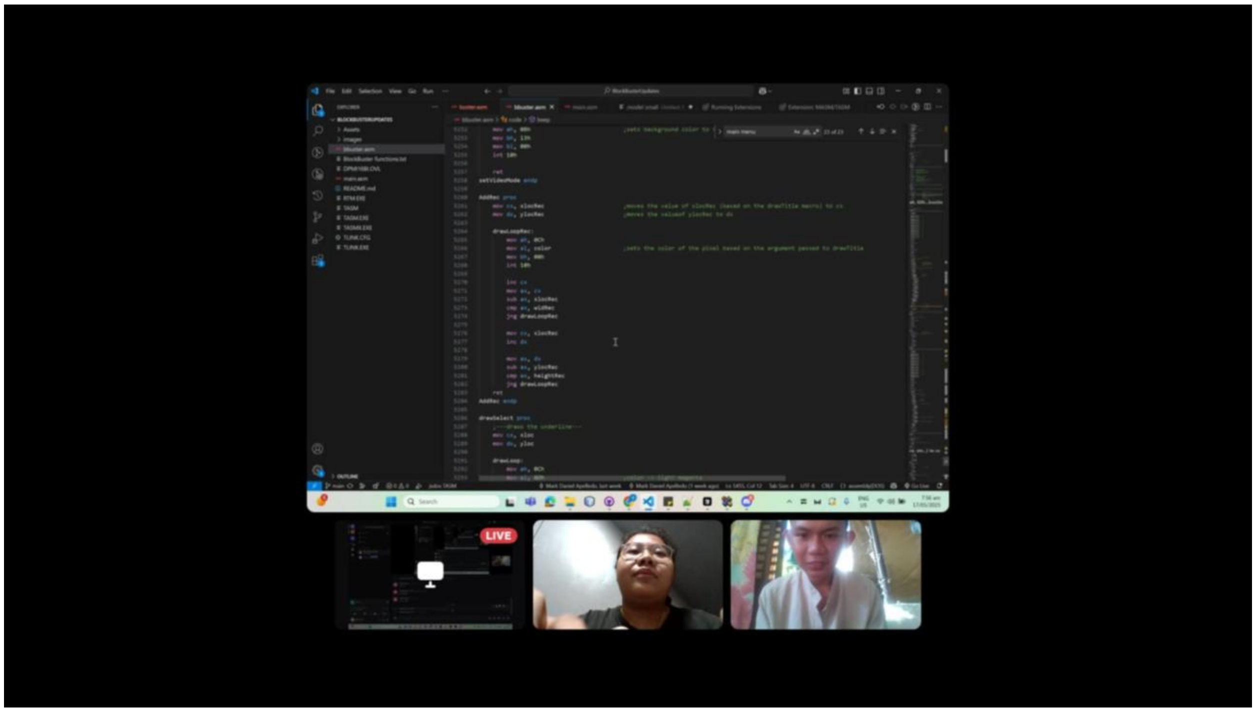



BLOCK
BUSTER


DOSBox-X 2025.02.01: BBBUSTER - 3000 cycles/ms
Main CPU Video Sound DOS Drive Capture Debug Help
BLOCK
BUSTER
PRESS [P] TO PLAY
PRESS [C] TO VIEW CONTROLS



```
    mov ah, 0Bh                    ;sets background color to (
    mov bh, 13h
    mov bl, 00h
    int 10h

    ret
setVideoMode endp

AddRec proc
    mov cx, xlocRec                ;moves the value of xlocRec (based on the drawTitle macro) to cx
    mov dx, ylocRec                ;moves the valueof ylocRec to dx

    drawLoopRec:
        mov ah, 0Ch
        mov al, color              ;sets the color of the pixel based on the argument passed to drawTitle
        mov bh, 00h
        int 10h

        inc cx
        mov ax, cx
        sub ax, xlocRec
        cmp ax, widRec
        jng drawLoopRec

        mov cx, xlocRec
        inc dx

        mov ax, dx
        sub ax, ylocRec
        cmp ax, heightRec
        jng drawLoopRec
    ret
AddRec endp

drawSelect proc
    ;---draws the underline---
    mov cx, xloc
    mov dx, yloc

    drawLoop:
        mov ah, 0Ch
```



```
    mov ah, 0Bh                    ;sets background color to (
    mov bh, 13h
    mov bl, 00h
    int 10h

    ret
setVideoMode endp

AddRec proc
    mov cx, xlocRec                ;moves the value of xlocRec (based on the drawTitle macro) to cx
    mov dx, ylocRec                ;moves the valueof ylocRec to dx

    drawLoopRec:
        mov ah, 0Ch
        mov al, color              ;sets the color of the pixel based on the argument passed to drawTitle
        mov bh, 00h
        int 10h

        inc cx
        mov ax, cx
        sub ax, xlocRec
        cmp ax, widRec
        jng drawLoopRec

        mov cx, xlocRec
        inc dx

        mov ax, dx
        sub ax, ylocRec
        cmp ax, heightRec
        jng drawLoopRec
    ret
AddRec endp

drawSelect proc
    ;---draws the underline---
    mov cx, xloc
    mov dx, yloc

    drawLoop:
        mov ah, 0Ch
```