University of Toronto

MIE438: Final Project Report
Group 18

Group Members

Mariem Ahmed

Tao Zhong

Qihan Gao

Victor Spiewak

Date of Submission: April 15tht, 2021

1. Introduction

1.1. Overview

The team decided to go for a Virtual Project. The project was to create a two-player Chess Game for the GameBoy Advance (GBA), played on an emulator like the VisualBoy Advance.

The main components that are used for this project are CPU - ARM7TDMI, Emulator (VisualBoyAdvance), and DevKitArm compilation environment. In this report, we will outline the basic hardware of GameBoy Advance which includes Memory Locations (VRAM and IO RAM), Graphics Hardware, Keypad Input Register and Hardware Interrupts. Each component will be explained in the GameBoy Advance Hardware and Specifications section 2 of the report.

The project is separated into several smaller subproblems which will be explained in the Detailed Final Design Section 3, this includes setting up a user interface to display the board, constraining the moves according to rules, and ultimately creating a two-player versus each other feature.

1.2. Change from Proposal

In the Design Proposal, the team originally planned to have a player-versus-player (PvP) mode as well as a player-versus-computer (PvC) mode for the chess game. However, as we moved forward in the course and consulted with the teaching team, the team found out that including a PvC mode was too software-oriented considering the scope of the course. As a result, the team decided to focus on PvP mode which fits better with the scope of the course. Any other design choices, including choice of hardware and virtual platform, remain unchanged.

2. GameBoy Advance (GBA) Hardware and Specifications

The Emulator is used to run the GameBoy Advance (GBA) game which is the VisualBoyAdvance, it provides a full debugging suite to help in testing and debugging our game while we are working on it, and is true to the original hardware of the GBA, allowing realistic emulation of the device virtually. The CPU - ARM7TDMI processor features, including a 32-bit RISC engine, 16-bit Thumb instruction set (smaller code size), debug functions, multiplier, and embedded ICE support logic [1]. DevKitArm is used to build and compile the C code (and ARM assembly code) into machine code for the GBA in a .gba ROM (Read-only Memory) format that the emulator can read and run [2]. It supports the ARM7TDMI's 32bit ARM and 16-bit Thumb instructions [2]. Figure 1, shows the Hardware Structure of the Game Boy Advance.

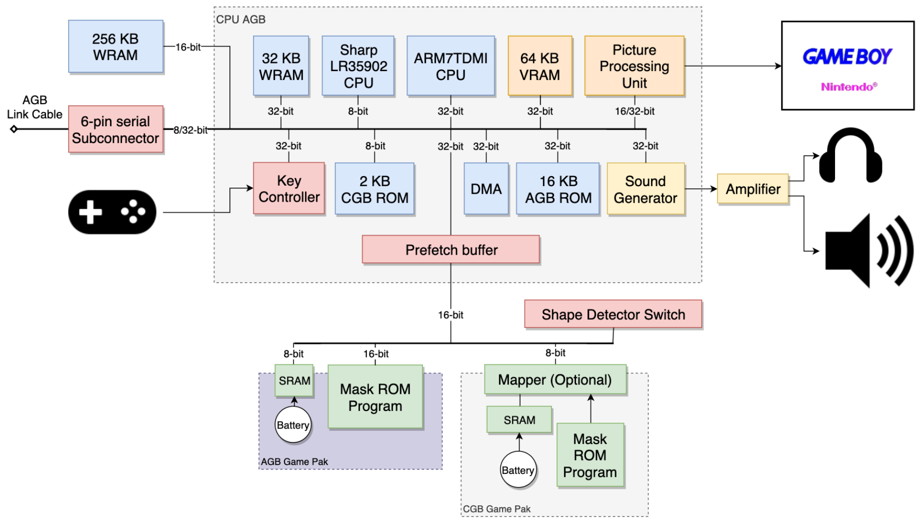


Figure 1: Overview of the Game Boy Advance Controller Hardware Structure [3]

2.1. Memory Locations

The team will deal with VRAM for drawing the screen and IO RAM for setting up the graphics and table X outlines the description of each memory used for this project. A detailed memory table will be in Appendix X. Figure X, shows the overview of the Memory Locations Structure in GBA.

Table X: General Memory Locations of GBA

Area	Description
IO RAM	Memory-mapped IO registers contain a mirror of the ASIC (Application Specific Integrated Circuit) registers on the GBA. This area of memory is used to control the graphics, sound, DMA, and other features. See memory-mapped IO registers for details on the function of each register [4].
VRAM	The video RAM is used to store the frame buffer in bitmapped modes, and the tile data and tilemaps for tile-based "text" and rotate/scale modes [4].

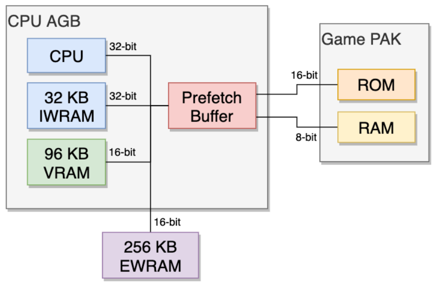


Figure 2: Memory Locations of the GBA [3]

2.2. Graphics Hardware

The GBA has a thin-film-transistor (TFT) colour LCD that is 240 x 160 pixels in size and has a refresh rate of exactly 280,896 CPU cycles per frame, or around 59.73 Hz [4]. Each consists of a 160 scanline vertical draw (VDraw) period followed by a 68 scanline blank (VBlank) period [4]. Furthermore, each of these scanlines consists of a 1004 cycle draw period (HDraw) followed by a 228 cycle blank period (HBlank) [4]. During the HDraw and VDraw periods the graphics hardware processes background and obj (sprite) data and draws it on the screen, while the HBlank and VBlank periods are left open so that program code can modify background and obj data without risk of creating graphical artifacts [4].

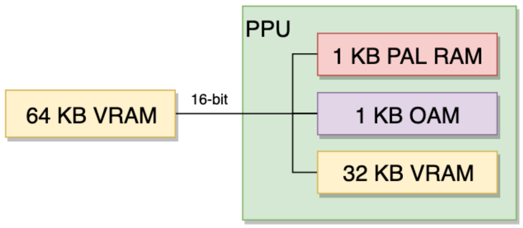


Figure 3: Memory Architecture of the Graphic Hardware in PPU [3]

2.3. Keypad Input Register

GBA has 4 direction keys, and 6 buttons: two control buttons (Select and Start); two regular fire buttons (A and B) and two shoulder buttons (L and R). This register stores the state of the GBA's buttons as shown in figure X. Each of the inputs is actively low [4]. This means that a '0' bit indicates that the key is pressed, while a '1' bit indicates that the key is not pressed [4].

Address: 0x4000130 - REG_KEY (The input register)(Read Only)

```
              R R  R R R R  R R R R
F E D C  B A 9 8  7 6 5 4  3 2 1 0
X X X X  X X J I  D U L R  S E B A

0 (A) = A button
1 (B) = B button
2 (E) = Select button
3 (S) = Start button
4 (R) = D-pad Right
5 (L) = D-pad Left
6 (U) = D-pad Up
7 (D) = D-pad Down
8 (I) = Right shoulder button
9 (J) = Left shoulder button
```

Figure 4: Key Input Register [4]

2.4. Hardware Interrupt

Interrupts are hardware flags that can be set to indicate that the team wants the CPU to stop whatever it is currently doing, to interrupt the program flow, and to execute something else when the interrupt has been handled to continue with the previous flow of the program. The team used the following interrupts in this project:

- V-Blank: Occurs when the V-count reaches 160, or 0xA0 [4].
- H-Blank: Occurs at the end of every raster line, from 0 - 228. H-blank interrupts occur during V-blank.
- V-Count: Occurs when the V-Count reaches the number specified in REG_DISPSTAT.
- Key: Occurs when the user presses or releases the buttons specified in REG_KEYCNT.

3. Detailed Final Design

This section outlines the GBA codes assembly used to build the Chess Game that includes the board game, chess pieces, get user input, logic check (Check legal moves, Check for Checkmate, and end game (winner)) and update chess pieces moves.

The following block diagram shows the overall flow of our main function.

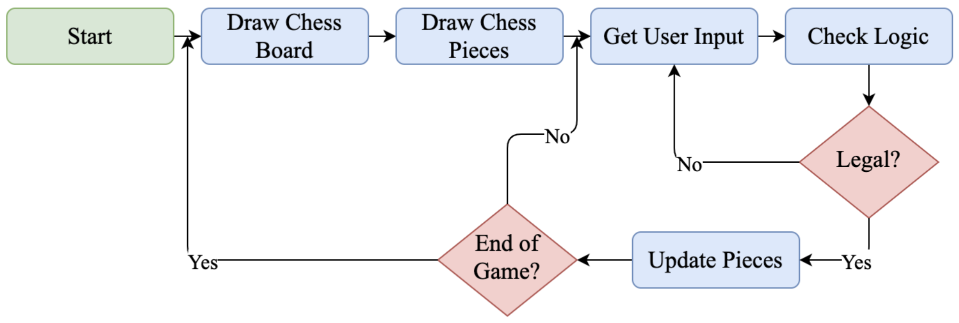
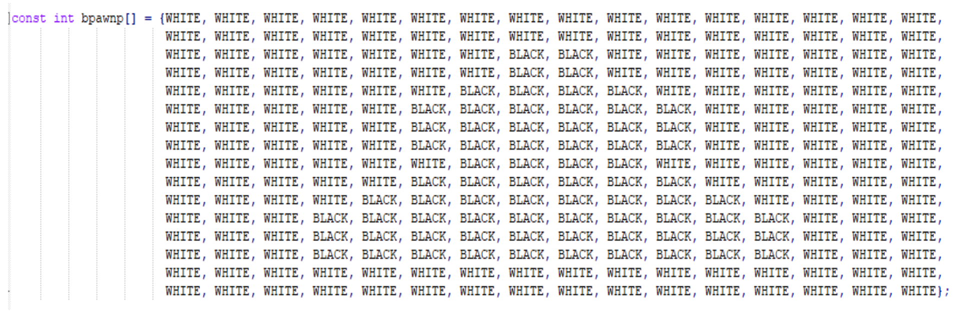


Figure 5: The overall flow of the program

The Draw Chessboard block initializes the Chessboard data structure and displays the board. In the Draw Chess Pieces block, the program initializes the 16 by 16 bitmap of the chess sprites and stores it. The bitmap will then be written into the VRAM as introduced in Section 2.1 so that it can be displayed.



```
const int bpawnp[] = {WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, BLACK, BLACK, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, BLACK, BLACK, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, BLACK, BLACK, BLACK, BLACK, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, WHITE, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, WHITE, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, WHITE, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, WHITE, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, WHITE, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, WHITE, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, BLACK, BLACK, BLACK, BLACK, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, WHITE, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, WHITE, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE,
                      WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE};
```

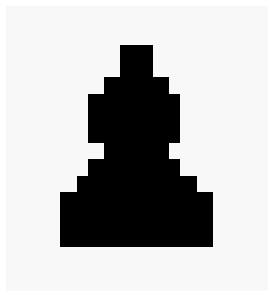


Figure 6: An example sprite of the black pawn in bitmap form

In terms of user input, the press of the joypad is constantly polled within the main loop via directly reading the Keypad Input Register as introduced in Section 2.3. The D-pad can control a cursor displayed in the user interface. The press of the A or B button can select or deselect a cell that the player wants to move. Then, the player can select a target cell. The whole process could be interpreted as a state machine as in the following diagram.

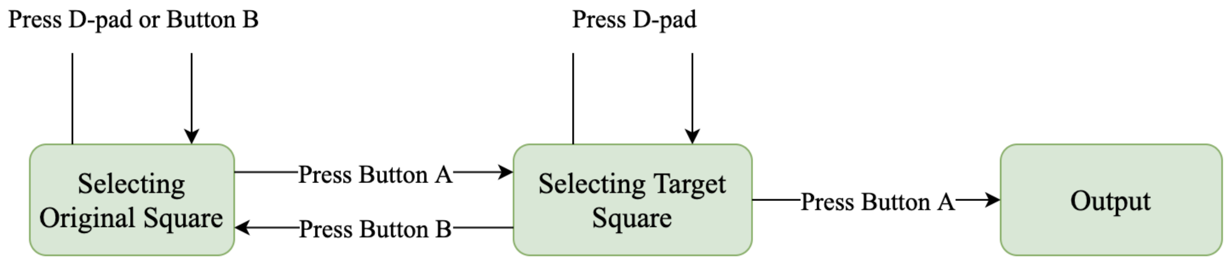


Figure 7: The logic of checking user input

After having the user input, the program will check whether the move is legal or not. If the move is illegal, the user needs to remake the move. This part is heavily software-oriented. However, unlike any other implementation of the chess game, the team made some modifications to the way to store the board state so that it is easy to not only check the logic but also update the display. Code could be found in our Github codebase [B1].

Lastly, we will update the pieces and board according to the move. The process is fairly simple. The original square will be drawn as an empty board square. The piece that was originally on the original square will be displayed on the target square. If the game comes to an end, the board will be reset.

4. Final Design Results

Following is a display of the graphics and the UI of the game. The chess pieces are drawn in colour of Black and White, and each piece has its unique sprite. The pieces were drawn with the bitmap described in Section 3 of the report.

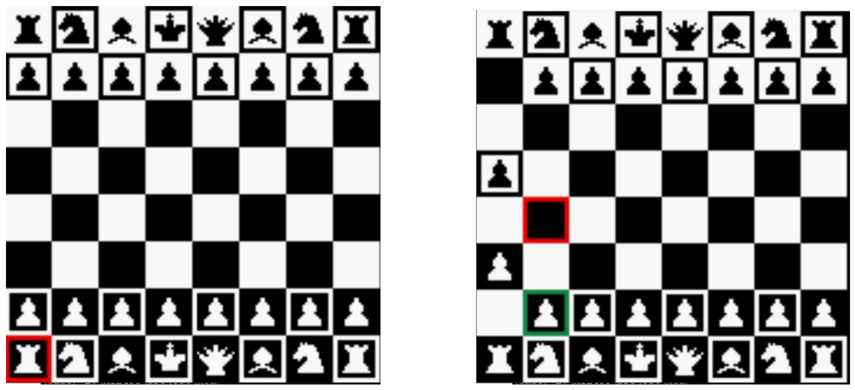
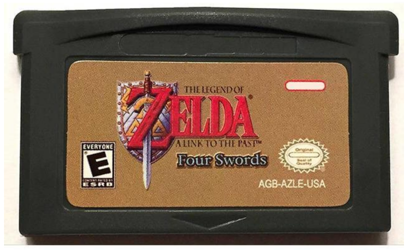


Figure 8: The initial setup of the game(left); Selecting a White Pawn to move(right)

The red box around the tile indicates the current location of the cursor. Once a piece is selected by the player pressing A, a green box is displayed around the piece to indicate the selected piece, then by moving the cursor around with the direction buttons, the player can select what move to take with the selected chess piece.

A Youtube link that shows the Chess Game displayed and played on GameBoy Advance is included in the following hyperlink, please click here [B3] for the video.

The final executable file is in the format of Chess.gba [B2], which can be run on all the GBA emulator platforms for playing. The demonstration and development of this project use Visual Boy Advance. In order to use it on a physical GBA machine, it needs to be loaded into a GBA game card.



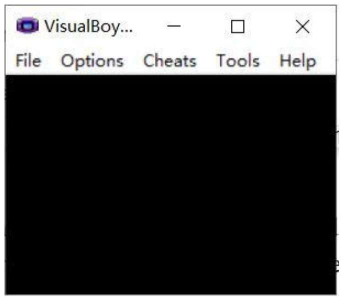


Figure 9: Physical GBA game card of The Legend of Zelda(left) [5]; The GBA emulator used, Visual Boy Advance(right)

In the current days, the Game Boy Advance is no longer being manufactured, using a physical platform to develop the game would be inconvenient and impractical. With a well-supported emulator, the team was able to fully simulate the game console while narrowing the scope of the project to focus on the embedded system instead of the electrical hardware, avoiding accommodating unnecessary processes such as loading the game into a game card and maintaining the condition of an old GBA machine. In addition to that, the team also benefited from the emulator's feature of video display when making demonstration videos. Recording from a physical GBA machine is a difficult process, while an emulator allows the convenient option of screen recording.

Using a virtual platform also simplifies the design process of the project by allowing each member of the team to be able to update and test the game code on their own computers. Given the current situation of COVID, the virtual platforms make the remote collaboration of the design team possible.

5. Recommendations

The objective of this project was to show the concept of playing a Chess Game using the GameBoy Advance (GBA). Moving forward, there are some areas of improvement that will make the game more complete, like the use of Sprites, Background and AI opponent.

5.1. Use the GBA's Built-in Sprites

As seen in *Figure 6*, each of the 6 chess pieces (Pawn, Rook, Knight, Bishop, Queen, and King) are stored as raw bitmapped values within the ROM (read-only memory) of the game, and then copied into VRAM to be displayed. The Game Boy Advance supports sprites, which are a more condensed version of bitmaps that are more memory and space-efficient than our current implementation. Take for example the Pawn, it is stored as a 256 int list, so over 1024 bytes of memory for only a single piece, and over 6kB for all the pieces. Sprites on the other hand allow each individual piece to be stored as a colour palette and bitmap. The colour palette is just mapping for the bitmap to each of the colours, so in our case, WHITE and BLACK, which would be encoded as 0, and 1 respectively. The bitmap could then be reduced to an 8 int list, each bit of memory representing a pixel and value 0 or 1, being the colour. This allows all the sprites to be stored in only 48 int, or 192 bytes, which is smaller than a single piece in our current implementation.

Another great feature of sprites is the abilities to update their location in VRAM automatically, allowing each sprite to have an X and Y position on the screen that can be changed when a piece moves, the GameBoy hardware will redraw the sprite in the correct location on the screen, as can be seen in the Figure below.

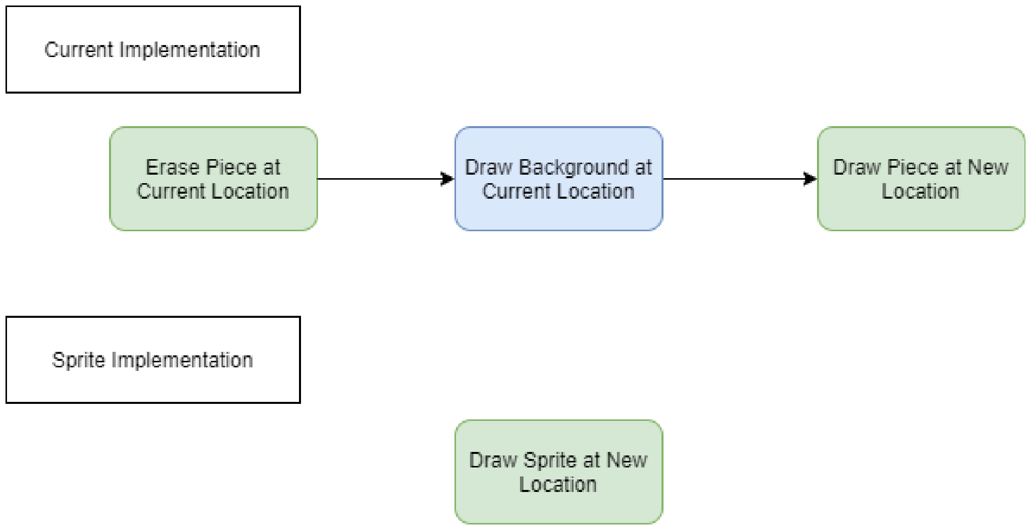

Current Implementation
Erase Piece at Current Location
Draw Background at Current Location
Draw Piece at New Location
Sprite Implementation
Draw Sprite at New Location

Figure 10: Comparison of the code flow using the Current Implementation vs Sprites

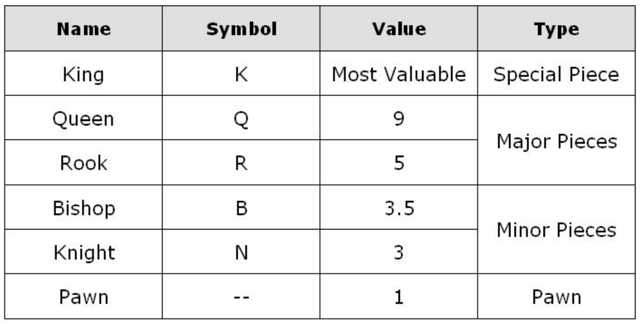
5.2. Use the GBA's Built-in Background

Similar to how sprites are used, a background can be set once at the beginning of the game (i.e. the chessboard) and wouldn't need to be constantly redrawn every time a piece moves, as seen in the figure above. This background implementation wouldn't save any memory, as the current implementation isn't storing the entire background in the ROM, rather determining the location on the chessboard where the piece was, and filling it entirely with the correct background colour; WHITE or BLACK. The major improvement would come in speed and efficiency as during each move the program is stuck in a loop that is 400 cycles long (number of pixels per chess space), which is a long time to spend in a loop only drawing, when the processor runs at 16.78MHz. This can lead to screen tearing (when the frame is updated before the processor is finished writing to VRAM) and cause other visual artifacts that detract from the gaming experience.

5.3. Add an AI Opponent

While the game can currently be played between two players, due to the COVID-19 pandemic and lack of online play, it makes for a lousy experience playing with two people on the same device (i.e. the emulator). The biggest improvement to the game would be the addition of an AI opponent for the user to play against. Due to the scope of the project, it was decided not to add an AI, however, there are several systems that need to be added to allow the addition of an AI into the game. While computers can easily beat even the most skilled chess players, the limited Gameboy hardware wouldn't be able to take advantage of the most sophisticated algorithms to play. The most basic AI would be one that would pick random legal moves each turn, not very effective and reasonably efficient since there are only at most 50 legal moves per turn, and on average 35. A very rudimentary AI would be trivial to include but no fun to play against since there is no strategy or logic in the movement.

A more nuanced approach that is very powerful but extremely computationally expensive would be to assign each chess piece a value, as in the figure below, and calculate the best move each turn based on the score increase from the move.



Name	Symbol	Value	Type
King	K	Most Valuable	Special Piece
Queen	Q	9	Major Pieces
Rook	R	5	
Bishop	B	3.5	Minor Pieces
Knight	N	3	
Pawn	--	1	Pawn

Figure 11: Value of chess pieces for a scoring algorithm

While this by itself is not that much more computationally expensive than a randomized naive approach, to get an AI that can match a person in skill, the algorithm would need to think beyond its next move, increasing the cost exponentially with the depth of the moves looked ahead.

6. References

[1] *Fujitsu.com*, 2021. [Online]. Available: https://www.fujitsu.com/cn/en/Images/ARMCore_Rev12-en.pdf. [Accessed: 30- Jan- 2021].

[2] "Getting Started - devkitPro", Devkitpro.org, 2021. [Online]. Available: https://devkitpro.org/wiki/Getting_Started. [Accessed: 30- Jan- 2021].

[3] "Game Boy Advance Architecture | A Practical Analysis", *Rodrigo's Stuff*, 2021. [Online]. Available: https://www.copetti.org/writings/consoles/game-boy-advance/. [Accessed: 14- Apr- 2021].

[4] *CowBite Virtual Hardware Spec*. [Online]. Available: https://www.cs.rit.edu/~tjh8300/CowBite/CowBiteSpec.htm#Memory-Mapped%20Hardware%20Registers. [Accessed: 15-Apr-2021].

[5] "The Legend of Zelda: Four Swords - GBA Game Card - Newegg.com", *Newegg.com*, 2021. [Online]. Available: https://www.newegg.com/p/01M-06AX-00026. [Accessed: 15- Apr- 2021].

7. Appendix

Appendix A: *General Memory Locations of GBA* [4]

Table 1: Memory Sections of GPA [4]

area	start	end	length	Port size	description
System ROM	0x00000000	0x0003FFF	16KB	32 bit	0x0 - 0x00003FFF contains the BIOS, which is executable but not readable.
EWRAM	0x02000000	0x0203FFFF	256KB	16 bit	**External work RAM** is available for the game's data and code. Though this is the largest area of RAM available on the GBA, memory transfers to and from EWRAM are 16 bits wide and thus consume more cycles than necessary for 32 bit accesses. Thus it is advised that 32 bit ARM code be placed in IWRAM rather than EWRAM.
IWRAM	0x03000000	0x03007FFF	32KB	32 bit	**Internal Work RAM** is fastest of all the GBA's RAM, being internally embedded in the ARM7 CPU chip package and having a 32 bit bus.
IO RAM	0x04000000	0x040003FF (0x04010000)	1KB	16 bit	**Memory-mapped IO** registers contain a mirror of the ASIC (Application Specific Integrated Circuit) registers on the GBA. It is used to control the graphics, sound, DMA, and other features.
PAL RAM	0x05000000	0x050003FF	1KB	16 bit	**Memory for two Palettes RAM** containing 256 entries of 15-bit colors each. The first is for backgrounds, the second for sprites.
VRAM	0x06000000	0x06017FFF	96KB	16 bit	**Video RAM** is used to store the frame buffer in bitmapped modes, and the tile data and tile maps for tile-based "text" and rotate/scale modes.
OAM	0x07000000	0x070003FF	1KB	32 bit	**Object Attribute Memory** is used to control the GBA's sprites.
PAK ROM	0x08000000	var	0 - 32 KB	16 bit	Game **Pak ROM** is where the game is located and execution starts. The size is variable, but the limit is 32 MB. If a cartridge is present on startup, the instruction found at location 0x08000000 is loaded into the program counter and execution begins from there.
Cart RAM	0x0E000000 or (0x0F000000)	var	0 - 64 KB	8 bit	**Cart RAM** is where saved data is stored.

Appendix B: Link to Github Repository and Youtube Video Demo

[B1] Link to Github Repository: https://github.com/VictorownzuA11/gba
[B2] Link to Compiled .gba output file:
https://github.com/VictorownzuA11/gba/blob/main/Chess.gba
[B3] Link to Youtube Video Demo: https://www.youtube.com/watch?v=bJDZ2MRgBuY