



עידו ברקן

215060104

י"ב 3

מורה מנחה: אופיר שביט

הגנת סייבר ומערכות הפעלה

תוכן עניינים

[מבוא 3](#_Toc121210985)

[הגדרת הפרויקט 3](#_Toc121210986)

[חקר שוק 4](#_Toc121210987)

[סקירת הפתרונות הקיימים 4](#_Toc121210988)

[השוואת הפרוייקט 5](#_Toc121210989)

[חקר פיתוחי 6](#_Toc121210990)

[איך אמולאטורים עובדים ונכתבים? 6](#_Toc121210991)

[הNES 8](#_Toc121210992)

[הרכיבים העיקריים של הNES 8](#_Toc121210993)

[מעבד 8](#_Toc121210994)

[מצבי הכתובות 10](#_Toc121210995)

[רכיב גרפי 12](#_Toc121210996)

[ספרייט 12](#_Toc121210997)

[גלילה 13](#_Toc121210998)

[רכיב שמע 14](#_Toc121210999)

[רכיבי קלט 14](#_Toc121211000)

[כרטיסי משחק 14](#_Toc121211001)

[Mappers - ממפים 15](#_Toc121211002)

[מה קורה כשאתה מכניס כרטיס משחק למערכת 16](#_Toc121211003)

[++C 17](#_Toc121211004)

[מה זה? 17](#_Toc121211005)

[יתרונות 17](#_Toc121211006)

[חסרונות 17](#_Toc121211007)

[הבחירה שלי בשפה זאת 17](#_Toc121211008)

[תוכנית העבודה של הפרויקט ובדיקות 18](#_Toc121211009)

[סדר הפרוייקט 18](#_Toc121211010)

[בדיקות 19](#_Toc121211011)

# מבוא

## הגדרת הפרויקט

שם הפרויקט: NESIDO - נסידו

הפרויקט יהווה אמולאטור (emulator) לקונסולה NES של Nintendo. בעזרת האמולאטור המשתמשים יוכלו להריץ מקבץ של משחקים שבנויים לקונסולה על המחשב האישי שלהם. הרעיון עצמו של אמולאטור הוא להשתמש במשאבים של המחשב המארח כדי לדמות את רכיבים של מערכת אחרת ובמקרה שלי המערכת האחרת תהיה ה NES.

כשחשבתי על רעיון לפרויקט חשבתי לעשות משהו בתחום מערכות ההפעלה שזה תחום שמעניין אותי והתחלתי לחפש על אפשרויות ודברים שאפשר לעשות, באחד מהחיפושים, נתקלתי בכתבה שסקרה את הרעיון של אמולאטור לקונסולה ישנה. התעניינתי מאוד ברעיון ולאחר חקר נוסף בו גיליתי על האתגרים המיוחדים ושילוב הנושאים שמעניינים אותי בתוך הפרויקט הזה, החלטתי שללא ספק אממש את זה כפרויקט הגמר שלי.

האתגר העיקרי שאני צופה בפרויקט הוא הסנכרון והתזמון של הרכיבים השונים שאני מדמה כדי שהמערכת תעבוד כגון: מעבד ורכיב גרפי. עצם העובדה שעוד לא יצא לי להתעסק בצורה כזאת רצינית בתחומי הlow של המחשב גם יהווה עבורי אתגר ויצריך חקר ולמידה.

המערכת מיועדת לכל אדם אשר חפץ בשימוש של קונסולת נס(NES) שכבר קשה מאוד ולא פרקטי למצוא ולהחזיק בימינו. אותו אדם יכול להיות ילד שמתעניין במשחקים של פעם או אדם מבוגר שרוצה להיזכר במשחקים ששיחק שהיה ילד.

המטרה העיקרית של הפרויקט היא קריאה והרצה של קוד משחק הבנוי לקונסולה על המחשב האישי של המשתמש בנוסף המערכת תכלול אפשרות בחירה בין משחקים שונים שאפשר להריץ עליה. התועלת המרכזית שהיא תיתן אפשרות לחוות שימוש בקונסולה בלי באמת להשתמש בקונסולה שאותה מאוד קשה להשיג.

## חקר שוק

### סקירת הפתרונות הקיימים

כיום יש מגוון אפשרויות לשימוש בקונסולת NES ואני אסקור את הפופולאריות ביותר בהשוואה למוצר שלי.

**NES**

אז האפשרות הראשונה היא כמובן למצוא את הקונסולה עצמה, לקנות ולהשתמש בה.



|  |  |
| --- | --- |
| **יתרונות** | **חסרונות** |
| * חווית המשתמש המקורית והאותנטית ביותר * המשחקים נוסו ונבדקו על הקונסולה עצמה ולכן רוב המשחקים ירוצו בצורה הטובה ביותר בלי בעיות על הקונסולה עצמה | * בימינו מאוד קשה להשיג אותה ואפילו אם מצאנו דרך להשיג אותה היא תהיה יקרה מאוד. * תופס מקום בבית, לא קומפקטי * מצריך שימוש במסך עם תאימות לחיבורים מיושנים שקשה להשיג. |

**Nestopia UE – אמולאטור לNES.**



|  |  |
| --- | --- |
| **יתרונות** | **חסרונות** |
| * תוכנה חינמית * כתובה ב C++ ולכן יעילה ומצליחה להריץ את המשחקים בצורה אופטימאלית. * בעלת ממשק משתמש – אפשרויות והגדרות כמו help ,load, sound, video | * המשתמש צריך לספק לבד את קבצי המשחק. |

**DOT Xyene**

|  |  |
| --- | --- |
| **יתרונות** | **חסרונות** |
| * תוכנה חינמית * בעלת תאימות להרצת מגוון רחב של משחקים – ממומש באמולאטור מספר גדול של mappers (יפורט בהמשך) | * המשתמש צריך לספק לבד את קבצי המשחק. * חסרת ממשק משתמש (גוררים קובץ משחק אל התוכנה והיא מריצה אותו) * כתובה בC# ולכן פחות יעילה, ייתכנו תקיעות במשחקים היותר מורכבים. |

### השוואת הפרוייקט

בדומה למערכות הקיימות הפרוייקט שלי יכלול ממשק משתמש בסיסי, ויאפשר הרצה של מגוון משחקים. המערכת גם תבטיח יעילות ומהירות והאמולאטור ייכתב בשפת C++. הפרוייקט יהיה חינמי לחלוטין. בנוסף לכך המערכת תספק למשתמש כבר מקבץ קטן של משחקים שהוא יוכל להריץ בלי לספק בעצמו את הקבצים.

# חקר פיתוחי

## איך אמולאטורים עובדים ונכתבים?

**רעיון בסיסי:**

אמולציה פועלת על ידי טיפול (דימוי) בהתנהגות המעבד והרכיבים של מערכת. בונים כל חלק של המערכת ואז מחברים את החלקים בדומה לחומרה.

**אמולציה למעבד:**

ישנן שלוש דרכים לטיפול באמולציית מעבד:

* פרשנות (Interpretation) - עם פרשנות, אתה מתחיל ב-IP (מצביע הוראות - נקרא גם PC, Program Counter) וקורא את ההוראה מהזיכרון. הקוד שלך מנתח הוראה זו ומשתמש במידע זה כדי לשנות את מצב המעבד. הבעיה המרכזית בפרשנות היא שזה מאוד איטי, בכל פעם שאתה מטפל בהוראה נתונה, עליך לפענח אותה ולבצע את הפעולה הנדרשת.
* קומפילציה דינמית מחדש - אתה עובר על הקוד בדומה לפרשנות, אבל במקום להפעיל אופקודים, אתה בונה רשימה של פעולות. ברגע שאתה מסיים קטע קוד מסויים, אתה מקמפל את רשימת הפעולות שהכנת לקוד מכונה עבור המחשב או מערכת המארחת , ואז אתה שומר את הקוד המקומפל ומבצע אותו. לאחר מכן, כאשר אתה שוב עובר על קטע קוד נתון, אתה רק צריך להפעיל את הקוד המקומפל מחדש מהזיכרון.
* קומפילציה סטטית מחדש - אתה עושה את אותו הדבר כמו בקומפילציה דינמית מחדש, אבל אתה עוקב אחר גושי קוד. בסופו של דבר אתה בונה גוש קוד שמייצג את כל הקוד בתוכנית, שאותו ניתן להפעיל ללא הפרעה נוספת. זה יהיה מנגנון נהדר אם לא היו בו את הבעיות הבאות:

1. קוד שלא נמצא בתוכנית מלכתחילה (למשל דחוס, מוצפן, נוצר/השתנה בזמן ריצה וכו') לא יעבור קומפילציה מחדש, ולכן הוא לא יפעל.
2. הוכח שמציאת כל הקוד בקוד בינארי נתון שווה ערך לבעיית העצירה (בקצרה: בהינתן תוכנית מחשב וקלט, האם התוכנית תסיים את פעולתה בשלב כלשהו עבור קלט זה).

שילוב בעיות אלה הופך את שיטת הקומפילציה הסטטית לבלתי אפשרי לחלוטין ב-99% מהמקרים.

לכל השיטות האלו, יש את אותה מטרה כוללת: לבצע קטע קוד כדי לשנות את מצב המעבד ולקיים אינטראקציה עם 'חומרה'. מצב המעבד הוא שילוב של אוגרי המעבד, מטפלי הפסיקות וכו'. עבור המעבד 6502 שמשתמשים בו בNES, יהיו מספר רגיסטרים בגודל 8 ביטים: A, X, Y, P ו-S יש גם רגיסטר אחד בגודל 16 ביטים שישמש בתור רגיסטר הIP (המצביע על הפקודה הבאה שאמורה להתבצע).

החלק השני של אמולציית מעבד הוא הדרך שבה אתה מתקשר עם החומרה. יש לזה בעצם שני צדדים:

תזמון מעבד - פלטפורמות מסוימות - במיוחד קונסולות ישנות יותר כמו הNES וכו' - מחייבות את האמולטור לתזמון קפדני כדי להיות תואם לחלוטין עם כל רכיבי המערכת. עם ה-NES, יש לך את ה-PPU (יחידת עיבוד פיקסלים – הרכיב הגרפי) שדורשת שה-CPU יכניס פיקסלים לזיכרון שלו ברגעים מדויקים. אם אתה משתמש בשיטת הפרשנות, אתה יכול בקלות לספור מחזורים (cycles) ובכך לדמות תזמון מתאים, עם קומפילציה דינמית/סטטית, הדברים הרבה יותר מורכבים.

טיפול בפסיקות(אינטראפטים) - הפסיקות הן המנגנון העיקרי שבעזרתו המעבד מתקשר עם החומרה. בדרך כלל, רכיבי החומרה יגידו למעבד מה הפסיקות שרלוונטיות אליהם. זה די פשוט - כאשר הקוד שלך זורק פסיקה נתונה, אתה מסתכל על טבלת הפסיקות וקורא לפעולה המתאימה.

**אמולציית לחומרה:**

יש שני דרכים לטיפול באמולציית התקן חומרה:

* הדמיית הפונקציונליות של המכשיר
* הדמיית ממשקי המכשיר בפועל

נקח את המקרה של הארד-דיסק.

דימוי הפונקציונליות יתבצע על ידי יצירת מקום אחסון, אפשרויות קריאה/כתיבה וכו'. חלק זה בדרך כלל פשוט מאוד.

הדמיית הממשק של המכשיר בפועל קצת יותר מורכב. בדרך כלל זהו שילוב כלשהו של רגיסטרים ממופי זיכרון ופסיקות

## הNES

NES – Nintendo Entertainment System היא קונסולת משחקי וידאו ביתית המיוצרת על ידי נינטנדו. היא שוחרר לראשונה ביפן ב-1983

לאחר שפיתחה סדרה של משחקי וידיאו מצליחים בתחילת שנות ה-80, נינטנדו תכננה ליצור קונסולת משחקי וידאו ביתית. נשיא נינטנדו הירושי ימאוצ'י דחה הצעות מורכבות יותר, וקרא לקונסולה פשוטה וזולה שמריצה משחקים המאוחסנים על כרטיסי משחק.

ה-NES הייתה אחת הקונסולות הנמכרות ביותר בתקופתה ועזרה להחיות את תעשיית המשחקים בארה"ב. ה-NES כלל מספר משחקים פורצי דרך, כמו Super Mario Bros ומשחקי The Legend of Zelda ו- Metroid, . בשנת 2011, IGN כינתה את ה-NES כקונסולת משחקי הווידאו הגדולה בכל הזמנים.



### הרכיבים העיקריים של הNES

#### מעבד

מעבד הNES מבוסס על מעבד 6502. ה-MOS Technology 6502 הוא מיקרו-מעבד של 8 ביטים ה-6502 הוא בעצם גרסה פשוטה, פחות יקרה ומהירה יותר של הMotorola 6800.

כשיצא בשנת 1975, ה-6502 היה המיקרו-מעבד הכי פחות יקר בשוק בהפרש ניכר. הוא נמכר בתחילה בפחות משישית מהעלות של עיצובים מתחרים מחברות גדולות יותר, כמו 6800 או אינטל 8080. הצגתו גרמה לירידות מחירים גדולות ומהירות בתמחור בכל שוק המעבדים. הוא עורר שורה של פרויקטים שהביאו למהפכת המחשבים הביתיים של תחילת שנות ה-80.

קונסולות משחקי וידאו ומחשבים ביתיים פופולריים של שנות ה-80 ותחילת שנות ה-90, כגון Atari 2600, Apple II, Commodore 64, Atari Lynx וגם הNES, משתמשים ב-6502 או וריאציות של העיצוב הבסיסי.

בקצרה, ה-6502 כולל:

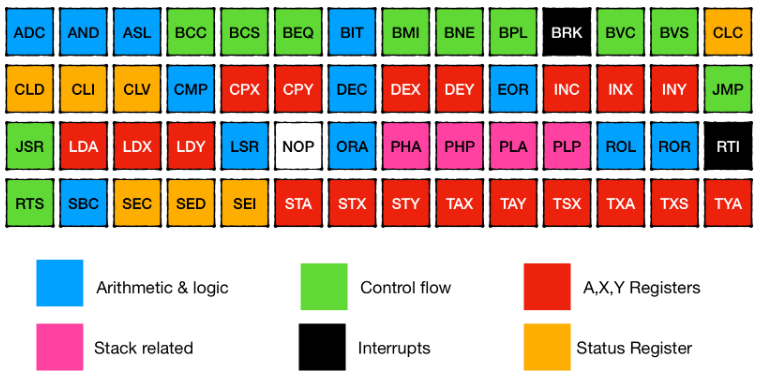
3 רגיסטרים למטרות כלליות בגודל 8 ביטים A, X ו-Y ומצביע מחסנית בגודל 8 ביטים.

רגיסטר אחד P אשר הביטים שלו מסמנים את מצב הדגלים.

מצביע מיקום שורת קוד הבאה (Program Counter) בגודל 16 ביטים.

ל-6502 יש 56 הוראות שונות אם כי חלקן מגיעות במגוון וריאציות באמצעות מצבי כתובת שונים, מה שהופך סה"כ 151 קודים חוקיים (אופקודים). ההוראות הן באורך של אחד, שניים או שלושה בתים, תלוי במצב הכתובת. הבית הראשון הוא ה-opcode והבתים הנותרים הם האופרנדים. ההוראות מתאימות למספר קבוצות:

* פעולות טעינה / אחסון - לטעון לרגיסטר ערך מהזיכרון או לאחסן את התוכן של רגיסטר בזיכרון.
* פעולות החלפה בין רגיסטרים- העתקת תוכן של רגיסטר X או Y לצובר A או העתק את התוכן של A לרגיסטרים X או Y..
* פעולות מחסנית - דחף או משוך מהמחסנית או תבצע פעלות על מצביע המחסנית בעזרת רגיסטר X.
* פעולות לוגיות - בצע פעולות לוגיות על הצובר A והזיכרון.
* פעולות אריתמטיות - בצע פעולות חשבון על רגיסטרים וזיכרון.
* הגדלות / הקטנות - הגדלה או הקטנה של רגיסטרים X או Y או ערך שמאוחסן בזיכרון.
* Shifts - העבר את הביטים של הצובר A או של מיקום זיכרון ביט אחד שמאלה או ימין.
* קפיצות / קריאות - הפסקת רצף קוד, וחידושו בכתובת שצוינה.
* הסתעפות - הפסקת רצף קוד, וחידושו מכתובת שצוינה, אם התנאי מתקיים. התנאי כולל בדיקת מצב הדגלים.
* פעולות רישום סטטוס – להדליק או לכבות דגל ברגיסטר הדגלים.



אריתמטיות ולוגיות

פעולות מחסנית

מערכת

הסתעפות וקפיצות

טעינה / אחסון

רישום סטטוס

המעבד הוא מעבד little endian כלומר הבית הפחות משמעותי בייצוג, מאוחסן בכתובת הקטנה יותר.

ל6502 יש מספר מצבי כתובת ((addressing modes – מצב כתובת מציין כיצד למצוא את כתובת הזיכרון של אופרנד כלומר זה אומר "מה מקור הנתונים המשמשים בהוראה זו?" נהוג לחלק את מצבי הכתובת ב6502 לשני סוגים, באינדקס ולא באינדקס.

##### מצבי הכתובות

**לא באינדקס ולא בזיכרון**

רוב מצבי הכתובת משתמשים בגישה לזיכרון, אך 3 לא.

אקומולטור(Accumulator) :

להוראות אלו יש את הרגיסטר A (אקומולטור) כפרמטר היעד ככה שלא צריך להוסיף עוד בתים.

מידי (Immediate):

להוראות אלו הפרמטרים שלהן הם קבועים המוגדרים כבית הבא אחרי ה-opcode.

מרומז (Implied):

בהוראה מרומזת, הנתונים ו/או היעד הם חלק מההוראה עצמה. לדוגמה, הוראת CLC משתמעת, היא הולכת לנקות את דגל ה-Carry של המעבד ולא מקבלת עוד נתונים.

**פעולות לא באינדקס עם הזיכרון**

יחסי (Relative):

כתובת יחסית משמשת להוראות הסתעפות (הוראות שתוצאתן גורם לקפיצה למקום אחר בקוד – שינוי ערך הPC). הוראות אלה לוקחות בית בודד, המשמשים כאופסט(offset) לקפיצה אם תקרה.

מוחלט (Absolute): כתובת מוחלטת מציינת את מיקום הזיכרון במפורש בשני הבתים שאחרי ה-opcode. כמובן בגלל שהמעבד הוא little endian אם נרצה להתייחס לדוגמה לכתובת $4032 עם האופקוד 4C זה ייראה כך : 4C 32 40.

זירו פייג' (Zero-Page):

לכל ההוראות התומכות בכתובת מוחלטת (למעט הוראות הJMP) יש גם אפשרות לקחת כתובת של בייט בודד. סוג זה של כתובת נקרא "Zero-Page" - רק העמוד הראשון (256 הבתים הראשונים) של הזיכרון נגיש. זה מהיר יותר, מכיוון שצריך לחפש רק בייט אחד.

עקיף (Indirect): הוראת JMP היא ההוראה היחידה שמשתמשת במצב פנייה זה. זוהי הוראה של 3 בתים - כתובת עקיפה משתמשת בכתובת מוחלטת כדי לחפש כתובת אחרת. כלומר הערך שנמצא במקום בזיכרון שאותו נותנים הוא יהיה הערך שאליו הPC ישתנה.

**פעולות באינדקס עם הזיכרון**

ל-6502 יש את היכולת לבצע כתובת אינדקסית, כאשר הרגיסטר X או Y משמש כאופסט נוסף לכתובת שניגשת אליה.

מוחלט באינדקס (Absolute Indexed):

מצב פנייה זה מכין את כתובת היעד על ידי הוספת התוכן של הרגיסטר X או Y לכתובת מוחלטת.

זירו פייג' באינדקס (Zero-Page Indexed):

זה עובד בדיוק כמו מוחלט באינדקס, אבל כתובת היעד מוגבלת ל256 הבתים הראשונים.

באינדקס עקיף (Indexed Indirect):

זה כמו שילוב בין זירו פייג' באינדקס ועקיף. אתה לוקח את הכתובת של זירו פייג', מוסיף לה את הערך של הרגיסטר X, ואז משתמש בזה כדי לחפש כתובת של שני בייטים.

עקיף באינדקס (Indirect Indexed):

עקיף באינדקס הוא כמו אינדקס עקיף אבל במקום להוסיף את אוגר X לכתובת לפני ההתייחסות, הכתובת של זירו פייג' יורדת, ואוגר Y מתווסף לכתובת המתקבלת.

הוראות עקיפות באינדקס הן 2 בתים - הבית השני הוא כתובת אפס עמודים - $86 בדוגמה. (לכן יש לאחסן את הכתובת שנלקחה בדף האפס.)

בעוד שהשימוש באינדקס עקיף מייצר רק כתובת של זירו פייג', כתובת היעד של מצב זה אינה עטופה - היא יכולה להיות בכל מקום במרחב הכתובות של 16 ביטים.



#### רכיב גרפי

NES PPU, או Picture Processing Unit, מייצר אות וידאו שמורכב עם 240 קווים של פיקסלים, שנועד להתקבל על ידי טלוויזיה. הNES בתחילת שנות ה-80, היא נחשבה למייצר תמונות דו-ממד מתקדם למדי עבור משחקי וידאו.

הוא עובר שוב ושוב על הזיכרון שלו ותפקידו לייצר פיקסלים כפלט, הוא מייצר אחד בכל פעם בהתבסס על תוכן הזיכרון והרגיסטרים שלו, וגם מגדיר רגיסטרי סטטוס כאשר אירועים שונים מתרחשים במהלך המעבר שלו. יש לו גם מרחב כתובות משלו, נפרד מזה של ה-CPU (למעשה שניים, שכן הנתונים של האובייקטים הזזים(sprites) נמצאים באזור נפרד משלו). הרגיסטרים שלו מספקים גם את הדרך היחידה למעבד לקרוא או לכתוב למרחב הכתובות של ה-PPU.

ל-NES יש פלטת צבעים המכילה 52 צבעים. עם זאת, לא ניתן להציג את כולם בזמן נתון. ה-NES משתמש בשתי פלטות, כל אחת עם 16 ערכים, פלטת התמונות ופלטת הספרייטים. פלטת התמונות מציגה את הצבעים הזמינים כעת עבור תמונות הרקע (האובייקטים שלא זזים). פלטת הספרייט מציג את הצבעים הזמינים כעת עבור ספרייטים. פלטות אלה אינן מאחסנות את ערכי הצבע בפועל אלא את האינדקס של הצבע בפלטת המערכת.

ל-NES יש שני טבלאות דפוס ב-$0000 ו-$1000. טבלאות הדפוס מאחסנות בלוקים של 8X8 פיקסלים שאותם שניתן לצייר על המסך.

בנוסף לטבלאות הדפוס יש גם טבלאות שמות שהן בעצם מטריצה של מספרי בלוקים, המצביעים על הבלוקים המאוחסנים בטבלאות הדפוסים. טבלאות השמות הן 32X30 פיקסלים ומכיוון שכל בלוק הוא 8x8 פיקסלים, כל טבלת שמות היא 256240x פיקסלים. לכל טבלת שמות יש טבלת תכונות משויכת. טבלאות תכונות מכילות את שני הביטים הגבוהים של הצבעים עבור הבלוקים. כל בייט בטבלת התכונות מייצג קבוצת בלוקים בגודל 44x, כך שטבלת תכונות היא טבלה בגודל 88x של הקבוצות הללו.

##### ספרייט

ספרייטים הם אובייקטים שאותם ניתן לצייר על המסך. ספרייטים יכולים להיות 8x8 פיקסלים או 8x16 פיקסלים. רוב הדמויות מורכבות ממספר ספרייטים. נתוני הספרייט מאוחסנים בטבלאות הדפוס. יש מקסימום 64 ספרייטים, שכל אחד מהם משתמש בארבעה בתים. הבתים פועלים באופן הבא:

•בית 0 - מאחסן את קואורדינטת ה-y של החלק השמאלי העליון של הספרייט מינוס 1.

• בית 1 - מספר אינדקס של הספרייט בטבלאות הדפוסים.

• בית 2 - מאחסן את התכונות של הספרייט.

• ביטים 0-1 - שני ביטים משמעותיים ביותר של הצבע.

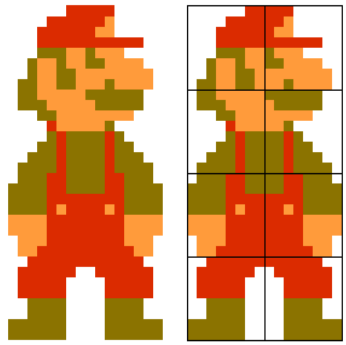
• ביט 5 - מציין אם לספרייט זה יש עדיפות על הרקע.

• ביט 6 - מציין אם להפוך את הספרייט אופקית.

• ביט 7 - מציין אם להפוך את הספרייט אנכית.

ספרייטים 8x16 פיקסלים משתמשים בטבלאות דפוסים שונות על סמך מספר האינדקס שלהם. אם מספר האינדקס הוא זוגי, נתוני הספרייט נמצאים בטבלת הדפוסים הראשונה ב-$0000, אחרת הם נמצאים בטבלת הדפוסים השנייה ב-$1000.

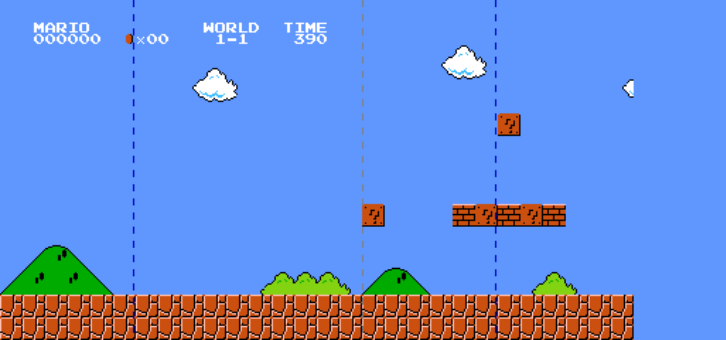
הדמויות בדרך כלל גדולות יותר מספרייט בודד ולכן נבנות על ידי שילוב ספרייטים מרובים. דוגמה המראה כיצד בנויה דמות מריו משמונה ספרייטים 8X8 נפרדים:



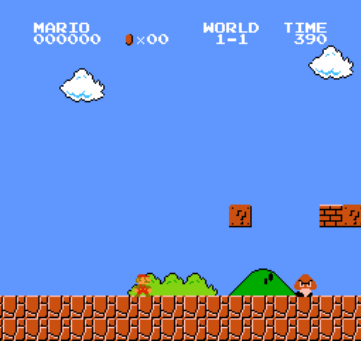
##### גלילה

לPPU יש יכולת נוספת עיקרית והיא הגלילה - ניתן לגלול את הרקע אופקית או אנכית. גלילה עושה שימוש בטבלאות השמות הנפרדות. בכל זמן נתון הרקע על המסך נלקח ישירות מאחת מטבלאות השמות או יהיה שילוב של שתי טבלאות שמות.

דוגמה לתוכן של שתיים מטבלאות השמות (הקו האפור מפריד בין שתי הטבלאות):



ודוגמה לתמונה המורכבת המוצגת על המסך, כולל ספרייטים:



#### רכיב שמע

NES APU הוא יחידת עיבוד השמע בקונסולת NES אשר תפקידה לייצר סאונד למשחקים.

#### רכיבי קלט

ל-NES יש שתי יציאות בקר לשימוש כללי בחזית הקונסולה. מכשיר הקלט המרכזי שמשתמשים בו בNES הוא השלט



#### כרטיסי משחק

כל כרטיס נשא לפחות שני שבבי ROM (Read Only Memory) גדולים - ה-Character ROM (CHR ROM) וה-Program ROM (PRG ROM). הראשון אחסן נתוני גרפיקת וידאו של משחק, השני אחסן הוראות מעבד - קוד המשחק. (במציאות, כאשר כרטיס מוכנס הCHR Rom מחובר ישירות ל-PPU, בעוד PRG Rom מחובר ישירות ל-CPU). הגרסה המאוחרת יותר של המחסניות נשאה חומרה נוספת (ROM ו-RAM) הנגישה באמצעות מה שנקרא Mappers. זה מסביר מדוע משחקים מאוחרים יותר סיפקו משחקיות וגרפיקה טובים יותר באופן משמעותי למרות שפועלים על אותה חומרת קונסולה.



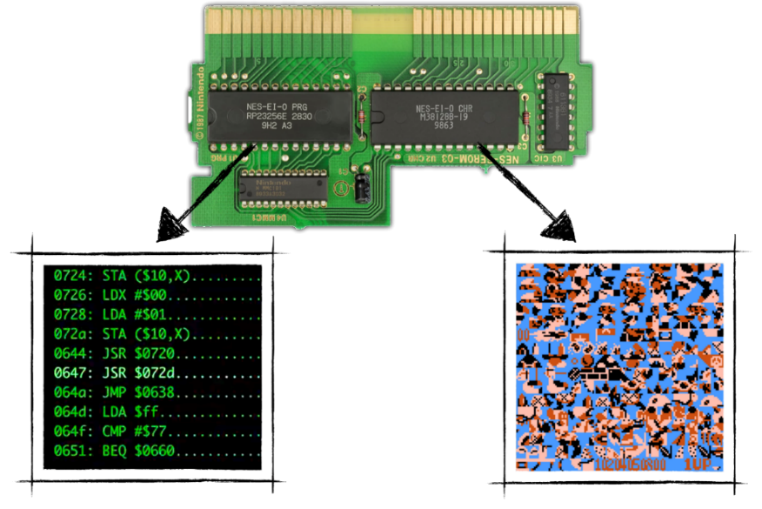
#### Mappers - ממפים

הזיכרון המוגבל של ה-NES הספיק למשחקים שיצאו בשנים הראשונות, אך ככל שהם הפכו מורכבים יותר, המשחקים הפכו גדולים יותר והזיכרון לא היה מספיק. כדי לאפשר לכרטיסי המשחק להכיל יותר ROM, ה-NES היה צריך להיות מסוגל להחליף את הנתונים בזיכרון ומחוצה לו כאשר היה צורך בכך. מכיוון שה-NES לא יכול היה לטפל מעבר לכתובת -FFFF, נעשה שימוש בחומרה בכרטיסים עצמם. חומרה זו הייתה ידועה בתור ממפה זיכרון או MMC (שבב ניהול זיכרון).

הרעיון הבסיסי של מיפוי הזיכרון הוא שכאשר המערכת דורשת גישה לנתונים במאגר ROM שאינו טעון כעת בזיכרון, התוכנה מצביעה על הצורך להחליף מאגרים והמאגר הנבחר נטען לזיכרון, ומחליף את התוכן הקיים. השימוש במפות זיכרון (mapper) היה אחד הגורמים שאיפשרו לNES לשרוד ליקויים טכנולוגיים.

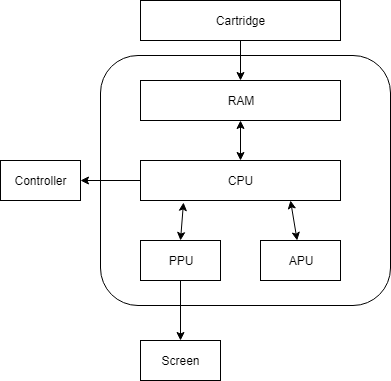
יש כ256 ממפים והם מסודרים לפי מספרים מהבסיסי עד המורכב (אחד מאוד בסיסי ופופולארי ו256 מאוד מורכב ונדיר במשחקים).

ההגבלה של המערכת שלי היא מגוון המשחקים שיהיה אפשר להריץ עליה. לחלק מהמשחקים שנבנו לNES נוספו גם mappers מורכבים במיוחד וספציפיים מאוד לאותו המשחק. במסגרת הפרויקט אני אממש טיפול בmappers הפופולארים ,הנפוצים והכלליים ביותר ולא בכולם ולכן לא יתאפשר להריץ משחקים מסוימים.



### מה קורה כשאתה מכניס כרטיס משחק למערכת

1. כרטיס משחק ממפה את תוכן ה-ROM שלה ל-RAM / מרחב כתובות של 16 ביטים.
2. המעבד מתחיל להריץ קוד
3. הקוד מחכה עד שה-PPU יאתחל
4. הקוד מתחיל לקרוא את הנתונים הגרפיים ולכתוב נתונים לתוך הPPU כדי להראות מסך ראשי.
5. הקוד מתחיל לקרוא את נתוני ה-ROM שלו וכותב לרגיסטרים של APU - זה נותן לך את הצליל.
6. הקוד ממתין לקלט שלך כדי לבצע בחירה ולהתחיל במסך הראשי
7. ברגע שתלחץ על התחל, המשחק טוען יותר נתונים ל-PPU, מפעיל את לולאת המשחק הראשית ומגיב לקלט שלך בהתאם



כרטיס משחק

שלט

מסך

## ++C

### מה זה?

C++ היא שפת תכנות למטרות כלליות שנוצרה על ידי Bjarne Stroustrup בשנת 1979 כשיפור לשפת C. מכיוון שזו הרחבה של C, היא ידועה גם בשם C עם מחלקות. מאז הקמתה, C++ התרחבה משמעותית עם הזמן. בשפה יש אפשרות לתכנות מונחה עצמים, פונקציונליות וגנריות בנוסף לשימוש וניהול זיכרון דינמי על ידי המתכנת. היתרון הגדול ביותר של C++ הוא שהיא ניתנת להרחבה ומאפשרת למפתחים שליטה רבה על האופן שבו היישומים שלהם מנצלים משאבים ובכך ניתן להגיע לביצועים מאוד גבוהים.

### יתרונות

מונחה עצמים (OOP) - C++ היא שפת תכנות מונחה עצמים מה שאומר שהפוקוס העיקרי הוא על אובייקטים ומחלקות.

מהירות - כאשר מהירות היא מדד קריטי, C++ היא הבחירה המועדפת ביותר. זמן הקומפילציה והביצוע של תוכנית C++ מהירים בהרבה מרוב שפות התכנות למטרות כלליות.

תמיכה בספריות עשירות - לספריית התבניות הסטנדרטית של (STL) יש פונקציות רבות זמינות כדי לעזור בכתיבת קוד. לדוגמה, ישנם STLs עבור מבני נתונים שונים כמו טבלאות hash, מפות, סטים וכו'.

תמיכת מצביע - C++ תומך גם במצביעים שלעתים קרובות אינם זמינים בשפות תכנות אחרות.

קרוב יותר לחומרה - C++ קרוב יותר לחומרה מאשר רוב שפות התכנות לשימוש כללי. זה עושה את זה מאוד שימושי באותם אזורים שבהם החומרה והתוכנה מחוברים זה לזה, ויש צורך בתמיכה ברמה נמוכה ברמת התוכנה.

### חסרונות

החיסרון העיקרי של C++ הוא שזו שפת תכנות מורכבת מאוד ללמידה. אם אתה חדש בתכנות ואינך מכיר שפות תכנות אחרות מונחות עצמים אז זה יכול לקחת הרבה מאוד זמן להבין, מה שמגביל את מה שאתה יכול לעשות איתה מלכתחילה.

אחת התכונות המרכזיות שהופכות את C++ לשפת תכנות ייחודית היא השימוש במצביעים. למרות שהם מאוד שימושיים, הם יכולים לתפוס הרבה זיכרון מערכת וזה לא אידיאלי כשאתה עובד על פרויקטים גדולים. מצביעים יכולים להיות גם מסובכים להבנה, ושימוש לא נכון בהם יכול לגרום לכל המערכת שלך לקרוס או להתנהג בצורה מוזרה, וזה חיסרון גדול.

למרות ששליטה בניהול הזיכרון נתפסת כיתרון על ידי אנשים רבים, הצורך להקצות זיכרון באופן ידני באמצעות מצביעים יכול לגזול זמן רב. אם אתה לא רגיל לעשות את זה אז ניהול זיכרון ידני זה הוא חיסרון גדול של C++, מכיוון שאין מנגנון garbage collectorלסינון אוטומטי של נתונים מיותרים.

### הבחירה שלי בשפה זאת

לאחר מחקר וסקירה של שפות תכנות הגעתי למסקנה כי ++C היא השפה המתאימה ביותר למימוש הפרוייקט שלי, השילוב של מחלקות ואובייקטים עם קוד יעיל ומהיר חשוב בשביל שהאמולאטור יעבוד והמשחקים ירוצו בצורה חלקה והטובה ביותר. סיבה נוספת שבחרתי בשפה זאת היא שאני מכיר אותה ברמה מאוד בסיסית ותמיד עניין אותי ללמוד אותה בצורה יותר מעמיקה.

## תוכנית העבודה של הפרויקט ובדיקות

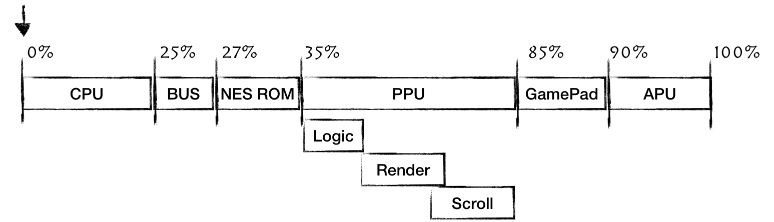
### סדר הפרוייקט

לאחר ביצוע מחקר במקורות שונים באינטרנט, הגעתי למסקנה שהסדר שאני אבחר ללכת איתו במימוש הפרוייקט הוא הסדר הבא:

( בין כל שלב אני אוסיף בדיקות עליהם ארחיב בהמשך)

1. **מימוש המעבד -** אתה לא יכול לעשות שום דבר בלי מעבד. ליישם את ההוראות ואת הלולאה הראשית.
2. **מימוש הROM (מידע שאותו מריצים – בדומה לכרטיסי המשחק) -** למה להוסיף תמיכת ROM לפני שאתה באמת יכול להציג משהו למסך? אני ארצה להשתמש בROMs בדיקה כדי לוודא שהדמיית המעבד שלי באמת טובה ולמצוא באגים מוקדם ככל האפשר. זה יחסוך לי המון זמן כיוון שהרבה יותר קשה למצוא את הבאגים במשחקים אמיתיים.
3. **מימוש הPPU -** זה כנראה הולך להיות השלב הכי ארוך ומאתגר. לוקח זמן להבין ולדמות את כל אופן הפעולה של הPPU. בשלב זה אני רק אדמה את אופן פעולת הPPU.
4. **מימוש עיבוד למסך -** עכשיו אני יכול להוסיף את העיבוד למסך ואת לולאת המשחק הראשית. בשלב זה אני לא אוסיף גלילה זה צריך להיות השלב הבא.
5. **מימוש הגלילה -** הגלילה היא מסובכת כי צריך לאתר את הפיקסל המדויק בתוך בלוק של 88x.
6. **הדמיית שימוש בשלט –** תמיכה בהקשות המקלדת.
7. **תמיכה לAPU –** הוספת פלט של צלילים.
8. **תמיכה בממפים (mappers) נוספים -** יותר ממפים = יותר משחקים. באופן כללי, הממפים הבסיסיים (הראשון, השני, השלישי...) הם אופציות מצוינות מכיוון שהן נתמכות על ידי רוב המשחקים.
9. **בניית ממשק משתמש**

בצורה מופשטת יותר:



**שלט**

**רכיב שמע**

**אופן פעולה**

**עיבוד למסך**

**גלילה**

**מעבד**

**זיכרון**

**רכיב גרפי**

### בדיקות

במסגרת הבדיקות אני אשתמש באתר nesdev.org שנותן מגוון ROMs לבדיקה עבור כל רכיב(מעבד, PPU, (APUברמות שונות של החמרה בבדיקה.

לאחר השלב הראשון אני אשתמש בבדיקה אחת מהאתר שבמקרה אצלה אני אקח את כל קוד הROM שלה ואכניס לתוך הקוד מכיוון שזה לפני מימוש הROM.

לאחר השלב השני אני אשתמש בכל הבדיקות הנמצאות באתר, בכל הROMs לבדיקה שניתנים – אני רוצה שהמעבד יעבוד בצורה הטובה ביותר ולא יהיו בו בעיות בכלל.

לאחר השלב השלישי אני אקח ROM אחד או שתיים מהאתר שאני אוכל שהמידע הגרפי נשמרת טוב בזיכרון (בשלב הזה עוד לא יהיה לי את הפלט למסך ובגלל זה אני ארצה לבדוק שקודם כל המידע הגרפי שהושג הוא מה שצריך להיות)

לאחר השלב הרביעי אני אוודא שתהליך לולאת המשחק תעבוד במחזורים מדויקים כמו החומרה האמיתית על ידי ספירת טיקים שחלפו. ובנוסף לכך בשלב הזה אני אנסה כמה משחקים שלא משתמשים בגלילה כגון דונקי קונג.

לאחר השלב החמישי אני אלך לקרוא את מסמך הגלילה הרשמי שפורסם על ידי Nintendo ואוודא את כל האינטראקציה בין רגיסטרי הPPU ופרמטרי הגלילה נעשים בדיוק כמו שצריך, לאחר מכן אני אנסה משחקים שמשלבים גלילה כמו Ice Climber.

לאחר השלב השישי אני אקח בדיקות מהאתר ואראה שהצלילים עובדים כראוי.

ולבסוף אחרי השלב השביעי אני אנסה להריץ את המשחקים ה"מורכבים" מתוך הרשימה שנמצאת באתר nesdev.org.