AMOS in Your Web Browser

Henrique de Freitas Serra

Yann-Gaël Guéhéneuc

Ptidej Team Meeting

25/07/16

Université Concordia University

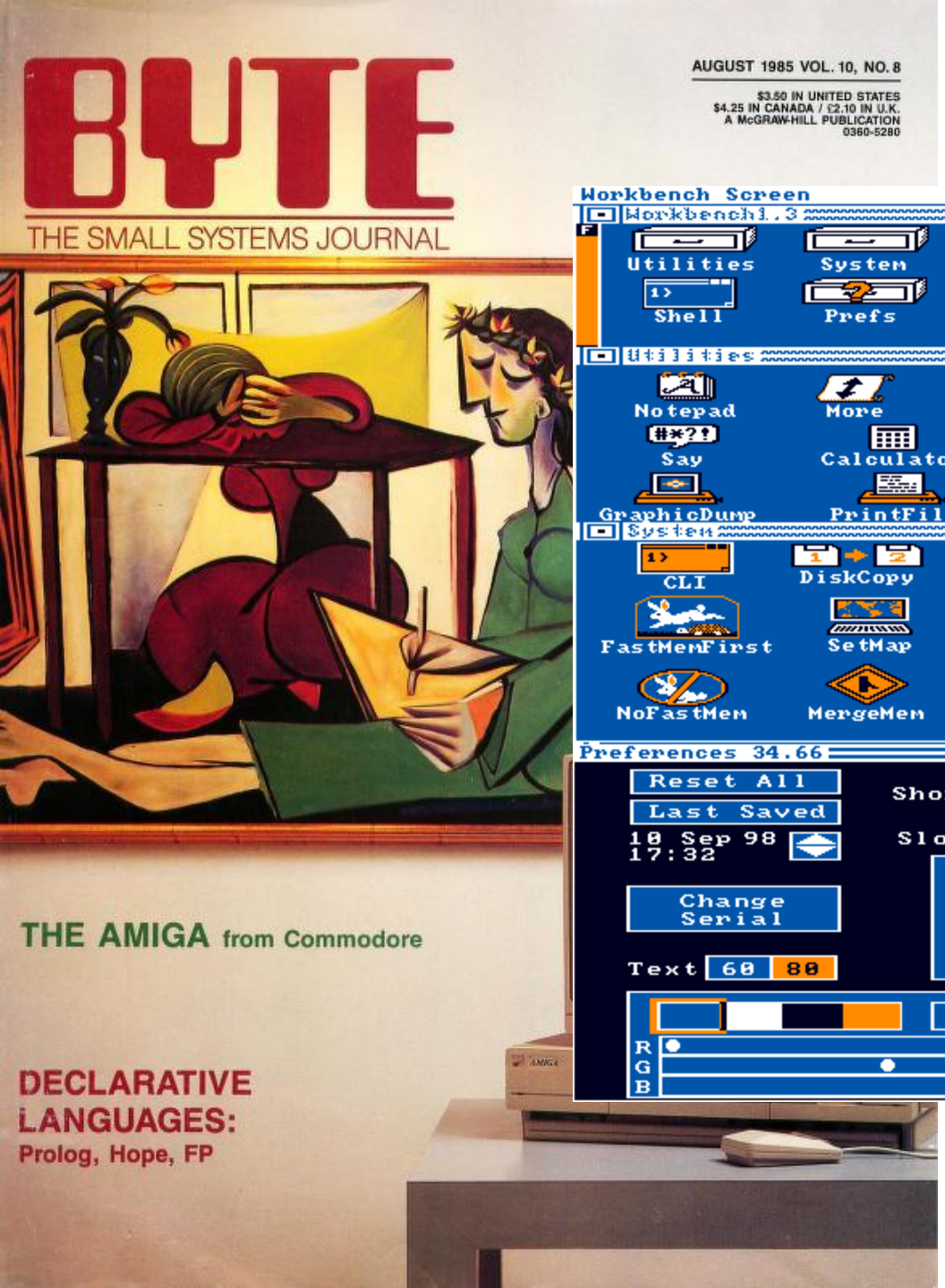
GINA CODY SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

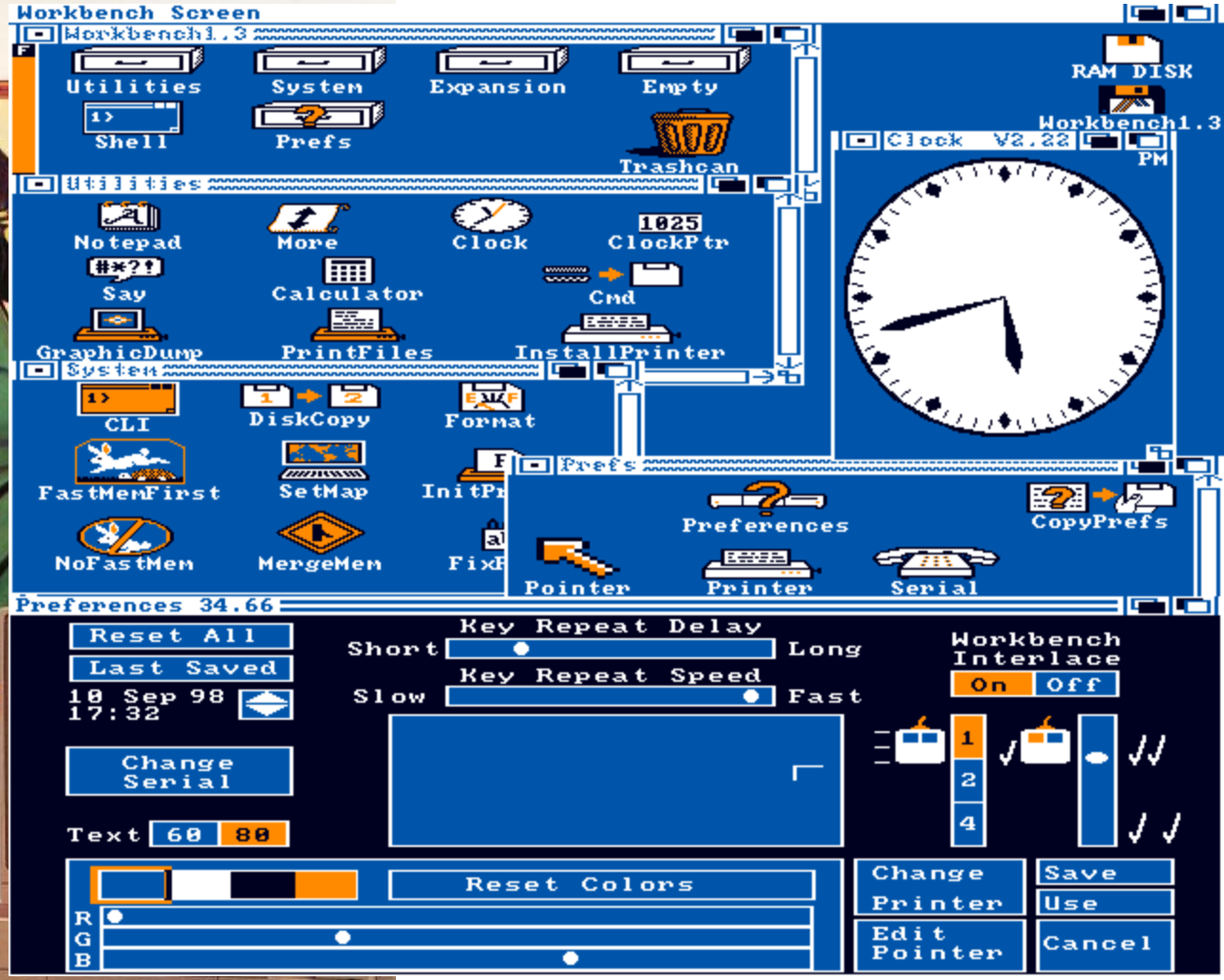
Department of Computer Science & Software Engineering

- I gave a previous version of this presentation at ReAnimate’25 on 25/06/12

Context

- The **Amiga computers**
  - Developed by Commodore
  - Launched in 1985 (with Andy Warhol!)
  - Groundbreaking computers at the time
    - **Custom chipsets** (OCS, ECS, and AGA)
      - Memory access
      - Graphics
      - Audio
    - **True preemptive multitasking**
    - **Graphical operating system**


BYTE
THE SMALL SYSTEMS JOURNAL
AUGUST 1985 VOL. 10, NO. 8
$3.50 IN UNITED STATES
$4.25 IN CANADA / £2.10 IN U.K.
A McGRAW-HILL PUBLICATION
0360-5280
THE AMIGA from Commodore
DECLARATIVE LANGUAGES:
Prolog, Hope, FP


Workbench Screen
Workbench1.3
Utilities
System
Expansion
Empty
Shell
Prefs
Trashcan
RAM DISK
Workbench1.3
Clock V2.22
PM
Utilities
Notepad
More
Clock
1025
ClockPtr
Say
Calculator
Cmd
GraphicDump
PrintFiles
InstallPrinter
System
CLI
DiskCopy
Format
FastMemFirst
SetMap
NoFastMem
MergeMem
Prefs
Preferences
CopyPrefs
Pointer
Printer
Serial
Preferences 34.66
Reset All
Last Saved
10 Sep 98
17:32
Change Serial
Text 60 80
Key Repeat Delay
Short
Long
Key Repeat Speed
Slow
Fast
Workbench Interlace
On
Off
1
2
4
Reset Colors
R
G
B
Change Printer
Edit Pointer
Save
Use
Cancel

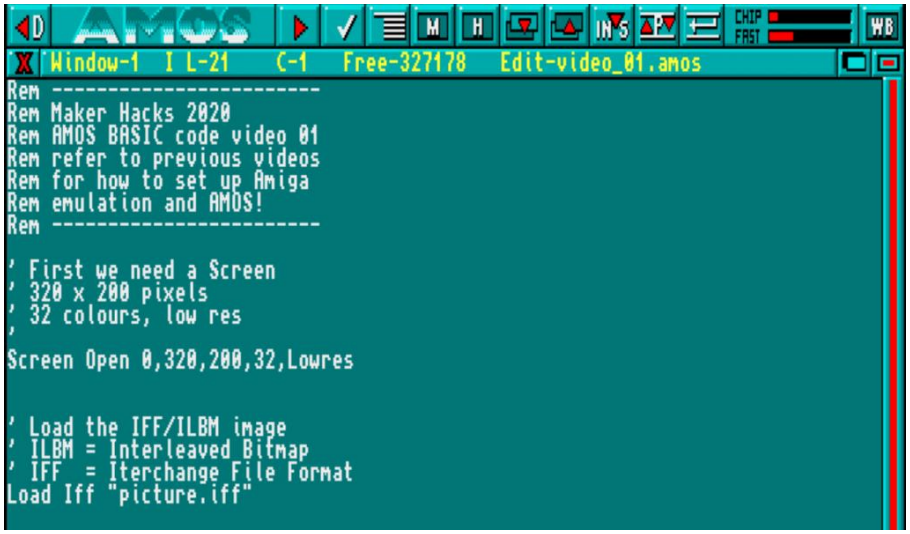
Commodore AMIGA 50

- The ultimate user-friendly personal computer.
- Simple and easy to use, it makes an excellent entry into both serious and entertaining computing.
- A personal computer designed for use in the home, school or small business.
- Ahead of its time: The Amiga 500 has set new standards in computing with its outstanding graphics, sound and animation capabilities.
- Versatile: Its multi-tasking syste can actually be conducting mo function at a time. Perfect tech increased productivity.
- **1084** High Resolution Colour
- Fully compatible with Amiga 5
- Genuine Commodore Product

Commodore

Context

- Programming on the Amiga
  - AmigaBasic
    - From Microsoft!
  - Assembly
  - C
    - SAS/C
    - StormC
    - GCC
    - vbcc
  - Even Java, more or less…



– One of the most popular programming languages on Amiga
– Released in 1990
– Designed for **ease of use**
  - **Games**
  - **Multimedia**
– Allowed creating graphical programs without (understanding) low-level programming

Problems

- Challenges
  - Running AMOS software today is **not trivial**
    - Requires **emulators** like **WinUAE** or **FS-UAE**
    - Replicate **complex chipset behavior**
      - Precise **video timings**
      - **Copper lists**, **Blitter operations**
    - Many programs are tightly coupled to the **original hardware**, making perfect emulation hard.

Problems

- And… the 100s commands
  - Each command is a method, definition, or instruction with which you can write code

```
Interface Instruction
Instruction
AMAL Function
Interface Function
```

```
Reserved Variable
Function
Embedded Menu Command
Structure
```

Solution

- Running AMOS in a Web browser
  - Translate and interpret AMOS into JavaScript
    - CRVJA: Compiler with Rules Validator to JavaScript from Amos
  - **Custom AMOS BASIC parser and JavaScript translator**, all running **directly in the browser**
    - Interpreter?
    - Transpiler?
    - Compiler?

Challenges

- Basic is basic **to write**
- AMOS Basic is complex **to interpret**
  - **826** command indexes
  - Which ones pertain to the hardware?
  - Which ones relate to graphics?
  - **Which of them are important?**

- Categories
  - **47** Interface Instructions
  - **22** Interface Functions
  - **442** Instructions
  - **222** Functions
  - **50** Structural Commands
  - **14** Reserved Variables
  - **14** AMAL Functions
  - **15** Menu Commands

Challenges

- Study each command and decide which one must be implemented
- Study each command to implement and decide how to translate it in JavaScript
  - If commands pertain to graphics, we will make them interact in the screen
  - If commands pertain to structure, we will change the behaviour of the JavaScript code

Challenges

- An environment must be created for commands to interact with one another
  - **E.g., Variable** commands used inside an **If** command
  - Must be **resolved by name** at runtime
    - If **x=10** was declared before some line of code, the transpiler must
      - Recognise that **X** exists
      - Insert `let x = 10;` earlier in the JavaScript output

Challenges

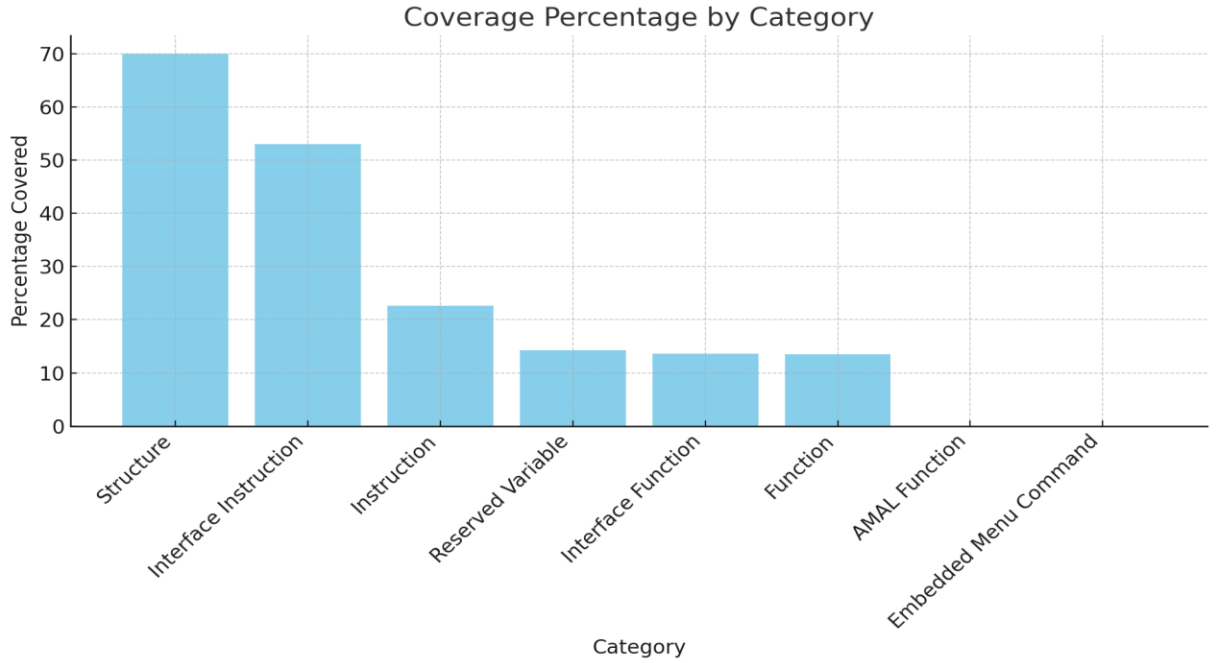
- Adding an **If** command inside a **For**
  - Each in their proper placement
  - Each with expected interactions
  - Not so Basic!

Current Progress

- **826** commands indexes
- **126** currently implemented

Current Progress

Category	Total	Covered	Percentage	Examples
Structure	50	35	70	IF, FOR, WHILE, REPEAT, PROC
Interface Instruction	47	25	53	CLS, INK, SCREEN OPEN
Instruction	442	100	22.6	PRINT, WAIT, SOUND, INK, TEXT
Reserved Variable	14	2	14.3	PI, TRUE, FALSE
Interface Function	22	3	13.6	EDIT INPUT, EDIT FIELD
Function	222	30	13.5	RND(), TIMER(), LEN(), MID$()
AMAL Function	14	0	0	(none)
Embedded Menu Command	15	0	0	(none)



Current Progress

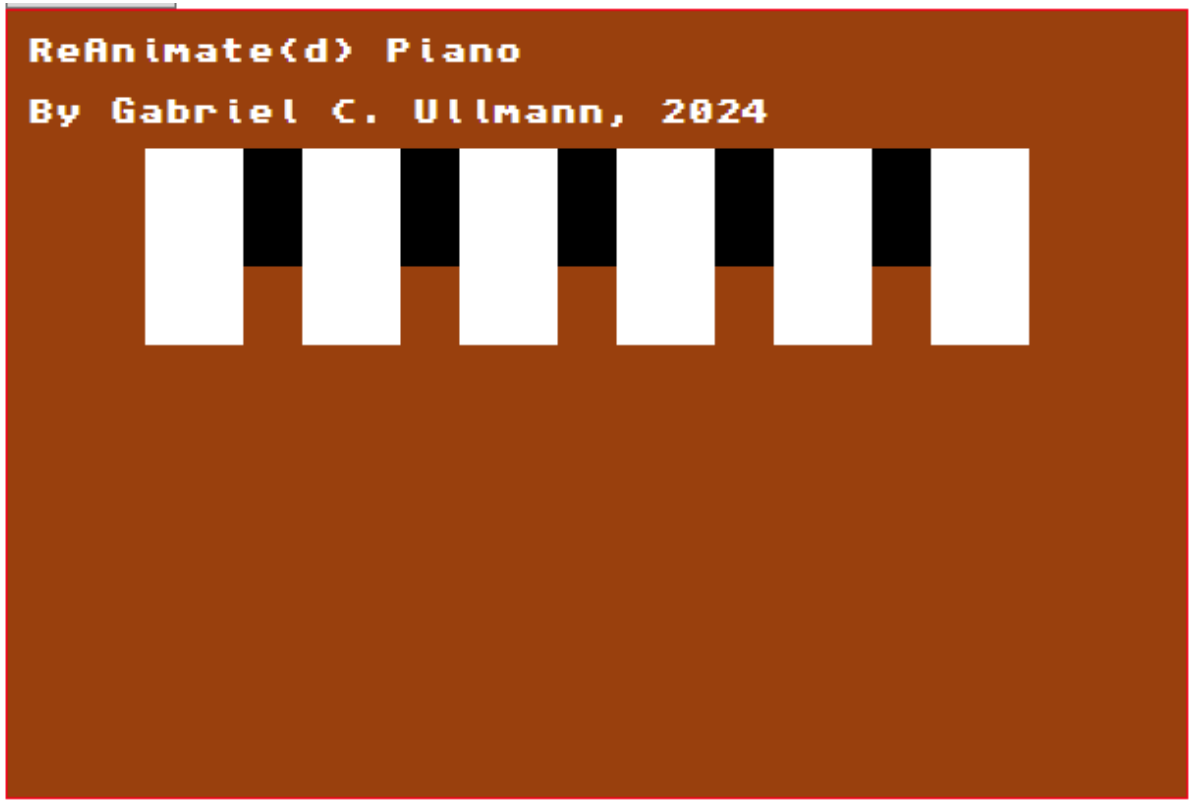
- All core commands implemented
  - Conditionals
  - Loops
  - Maths

```
expression1:
    term ((ADD | SUBTRACT) term)* NUMBER? // Handle addition and subtraction
    ;

term:
    SUBTRACT? factor ((MULTIPLY | DIVIDE) factor)* // Handle multiplication and division
    ;
array_index_get:
    IDENTIFIER BRACKETOPEN_PROP (expression1) BRACKETCLOSE_PROP
    ;
factor:
    NUMBER                    // A number
    | array_index_get
    | sin_function
    | cos_function
    | rndFunction
    | IDENTIFIER               // A variable
    | '(' expression1 ')'       // Parentheses for grouping
    | HEX_NUMBER

    ;
```

```
if_then:
IF expression1 expressions_comparators? expression2 (or_and expression1 expressions_comparators expression2)? 'then' statement
;
```

```
enterIf_then(ctx) {
  let expressions1 = [];
  let expressions2 = [];
  let comparators = [];
  let or_and = [];
  for (let i = 0; i < ctx.expression1().length; i++) {
    expressions1.push(ctx.expression1(i).getText());
  }
  for (let i = 0; i < ctx.expression2().length; i++) {
    expressions2.push(ctx.expression2(i).getText());
  }
  for (let i = 0; i < ctx.expressions_comparators().length; i++) {
    comparators.push(ctx.expressions_comparators(i).getText());
  }
  for (let i = 0; i < ctx.or_and().length; i++) {
    or_and.push(ctx.or_and(i).getText());
  }

  let finalIfStatement = "";

  for (let i = 0; i < expressions1.length; i++) {
    finalIfStatement +=
      expressions1[i] + " " + comparators[i] + " " + expressions2[i];
    if (or_and[i] && or_and[i] === "AND") {
      finalIfStatement += " && ";
    }
    if (or_and[i] && or_and[i] === "OR") {
      finalIfStatement += " || ";
    }
  }

  console.log(finalIfStatement);
}
```

Current Prog



- Graphics-relat
  - To display a p
    - Give x and y

```
Text 10,10,"ReAnimate(
Text 10,20,"By Gabriel
```

    - Parse the lines of code

```
text:
    TEXT NUMBER COMMA NUMBER COMMA (STRING | IDENTIFIER)
    ;
```

    - Generate HTML using JS

```
1275  enterText(ctx) {
1276    const x = ctx.children[1]?.getText();
1277    const y = ctx.children[3]?.getText();
1278    const text = ctx.children[5]?.getText();
1279    if (!text.includes('"')) {
1280      this.output += `
1281
1282  ${this.indent()}const textDiv${x}${y} = document.createElement('div');
1283  ${this.indent()}textDiv${x}${y}.innerText = ${text};
1284  ${this.indent()}textDiv${x}${y}.id = 'textDiv' + '${x}' + '${y}';
1285  ${this.indent()}textDiv${x}${y}.style.position = 'absolute';
1286  ${this.indent()}textDiv${x}${y}.style.left = '${x}px';
1287  ${this.indent()}textDiv${x}${y}.style.top = '${y}px';
```

Current Progress

- Audio-related commands
  - Set a map for pitch frequency **1** to **10** with **all notes**
  - Build a **wavelengt function/oscillato**
  - Simple sound
    - **Play 37+I,1**

```
const pitchToFrequency = {
    1: 16.35,   // C0
    2: 17.32,   // C#0
    3: 18.35,   // D0
    4: 19.45,   // D#0
    5: 20.60,   // E0
    6: 21.83,   // F0
    7: 23.12,   // F#0
    8: 24.50,   // G0
    9: 25.96,   // G#0
   10: 27.50,   // A0
   11: 29.14,   // A#0
```

```
let activeOscillators = {}; // Object to store active oscillators keyed by noteId

function soundPlayer(noteId, cooldown) {
    let currentTime = Date.now();

    if (currentTime - soundPlayerTimeTracker > cooldown/2) {
        soundPlayerTimeTracker = currentTime;

        const frequency = pitchToFrequency[noteId];

        // Check if there's already an oscillator for this noteId
        if (activeOscillators[noteId]) {
            // Stop the existing oscillator
            activeOscillators[noteId].stop();
            activeOscillators[noteId].disconnect(); // Disconnect it from the audio context
        }

        // Create a new AudioContext for the new oscillator
        const audioCtx = new (window.AudioContext || window.webkitAudioContext)();

        // Create a GainNode for controlling volume
        const gainNode = audioCtx.createGain();
        gainNode.gain.setValueAtTime(0, audioCtx.currentTime); // Start at zero gain (silent)

        // Create a new oscillator
        const oscillator = audioCtx.createOscillator();

        // Set the frequency of the oscillator
        oscillator.frequency.setValueAtTime(frequency, audioCtx.currentTime);

        // Create a custom waveform using PeriodicWave
        const real = new Float32Array([0, 1, 0.5, 0.25, 0.125]); // Amplitude of harmonics
        const imag = new Float32Array(real.length); // Zero phase shift
        const customWave = audioCtx.createPeriodicWave(real, imag);

        // Set the custom waveform to the oscillator
        oscillator.setPeriodicWave(customWave);

        // Connect the oscillator to the gain node, then to the audio context's destination (the speakers)
```

Current Progress

- Banks and the code encodings

Offset	Length	Description
0	4 bytes	ASCII identifier **AmBk**
4	2 bytes	bank number (1-15 for AMOS, 1-65535 for AMOS Pro)
6	2 bytes	flags ▪ bit 0 set means the bank can be loaded into either CHIP memory or FAST memory. ▪ bit 0 cleared means the bank must be loaded into CHIP memory.
8	4 bytes	bank length ▪ bits 27-0: length of bank data + 8. Subtract 8 to get true bank length ▪ bits 29-28: undefined ▪ bit 30: if set, *try* loading bank in CHIP memory (if that fails, FAST memory is OK) ▪ bit 31: if set, *try* loading bank in FAST memory (if that fails, CHIP memory is OK)
12	8 bytes	bank type: unterminated ASCII string which is padded with spaces
20	? bytes	bank data. What's here depends on the bank type, its length is given in the bank length field

Section		Length
Header identifying which version of AMOS saved the file • "AMOS Pro111V" and 4 more bytes (AMOS Professional, source tested) • "AMOS Pro111v" and 4 more bytes (AMOS Professional, source not tested) • "AMOS Pro101V" and 4 more bytes (AMOS Professional, source tested) • "AMOS Pro101v" and 4 more bytes (AMOS Professional, source not tested) • "AMOS Basic V134 " (AMOS Pro compatible with AMOS 1.3, source tested) • "AMOS Basic v134 " (AMOS Pro compatible with AMOS 1.3, source not tested) • "AMOS Basic V1.3 " (AMOS The Creator v1.3, source tested) • "AMOS Basic v1.3 " (AMOS The Creator v1.3, source not tested) • "AMOS Basic V1.23" (AMOS The Creator v1.2, source tested) • "AMOS Basic v1.23" (AMOS The Creator v1.2, source not tested) • "AMOS Basic V1.00" (AMOS The Creator v1.0 – v1.1, source tested) • "AMOS Basic v1.00" (AMOS The Creator v1.0 – v1.1, source not tested)		16 bytes
Length in bytes of tokenized BASIC code to follow		4 bytes
Tokenized BASIC code		varies
AMOS AmBs segment	ASCII identifier "AmBs"	4 bytes
	Count of AMOS banks to follow (0–16)	2 bytes
	AMOS banks. Each bank's length must be individually determined.	varies

Tokenised BASIC code

Tokenised BASIC code is a sequence of tokenised lines. Each tokenised line has the following format:

Field	Length
Length of this line in words (2 bytes), including this byte. To get the length of the line in bytes, double this value	1 byte
Indent level of this line. Prefix *indent level* + 1 spaces at the beginning of the line, or no spaces if the value is less than 2	1 byte
Sequence of tokens. Each token is at least two bytes, and all tokens are rounded to to a multiple of two bytes. Each token is individually sized. The tokens always end with a compulsory null token	varies

Current progress

- Favorite ones
  - Sprite Banks
  - Special tokens

Offset	Length	Description
0	4 bytes	ASCII identifier **AmSp** (sprites, load to bank 1) or **AmIc** (icons, load to bank 2)
4	2 bytes	the number of sprites/icons to follow
6	? bytes	sprite/icon data. Each sprite/icon is individually sized and has this format:
6+?	64	a 32-entry colour palette. Each entry has the Amiga COLORx hardware register format, $0RGB

Offset	Length	Description
0	2 bytes	width of the sprite/icon, in 16-bit words rather than pixels (*w*)
2	2 bytes	height of the sprite/icon, in pixels (*h*)
4	2 bytes	depth of the sprite/icon, in bitplanes (1-5)
6	2 bytes	hot-spot X co-ordinate
8	2 bytes	hot-spot Y co-ordinate
10	*w**2**h***d* bytes	planar graphic data: plane 0 data first, then planes 1, 2, 3, 4 if present

Token	Type	Interpretation
0x064A	`Rem`	• 2 bytes: token (0x064A or 0x0652) • 1 byte: unused • 1 byte: length of ISO-8859-1 string to follow • variable length: ISO-8859-1 string, with the above-given length. The string is null terminated and its length is rounded up to a multiple of two. The string should be printed after the remark token.
0x0652	`'`	
0x023C	`For`	• 2 bytes: token • 2 bytes: unknown purpose
0x0250	`Repeat`	
0x0268	`While`	
0x027E	`Do`	
0x02BE	`If`	
0x02D0	`Else`	
0x0404	`Data`	
0x25A4	`Else If`	
0x0290	`Exit If`	• 2 bytes: token • 4 bytes: unknown purpose
0x029E	`Exit`	
0x0316	`On`	

0x0376	`Procedure`	• 2 bytes: token
		• 4 bytes: number of bytes to corresponding `End Proc` line
		(start of line + 8 + above = start of `End Proc` line)
		(start of line + 8 + 6 + above = line after `End Proc` line)
		• 2 bytes: part of seed for encryption
		• 1 byte: flags
		  • bit 7: if set, procedure is folded
		  • bit 6: if set, procedure is locked and shouldn't be unfolded
		  • bit 5: if set, procedure is currently encrypted
		  • bit 4: if set, procedure contains compiled code and not tokens
		• 1 byte: part of seed for encryption

0x2A40	`Equ`	• 2 bytes: token • 4 bytes: stored equate value • 1 byte: equate type (0-7) • 1 byte: unknown purpose
0x2A40	`Lvo`	
0x2A54	`Struc`	
0x2A64	`Struct`	

Current Progress

- Implementation and testing
  - Many code samples
  - Encoded in the AMOS file format
    - File extension `.amos`
  - AMOS files are parsed by the AMOS interpreter
  - Decoder in CRVJA to directly read AMOS files
    - Binary specifications of the file format

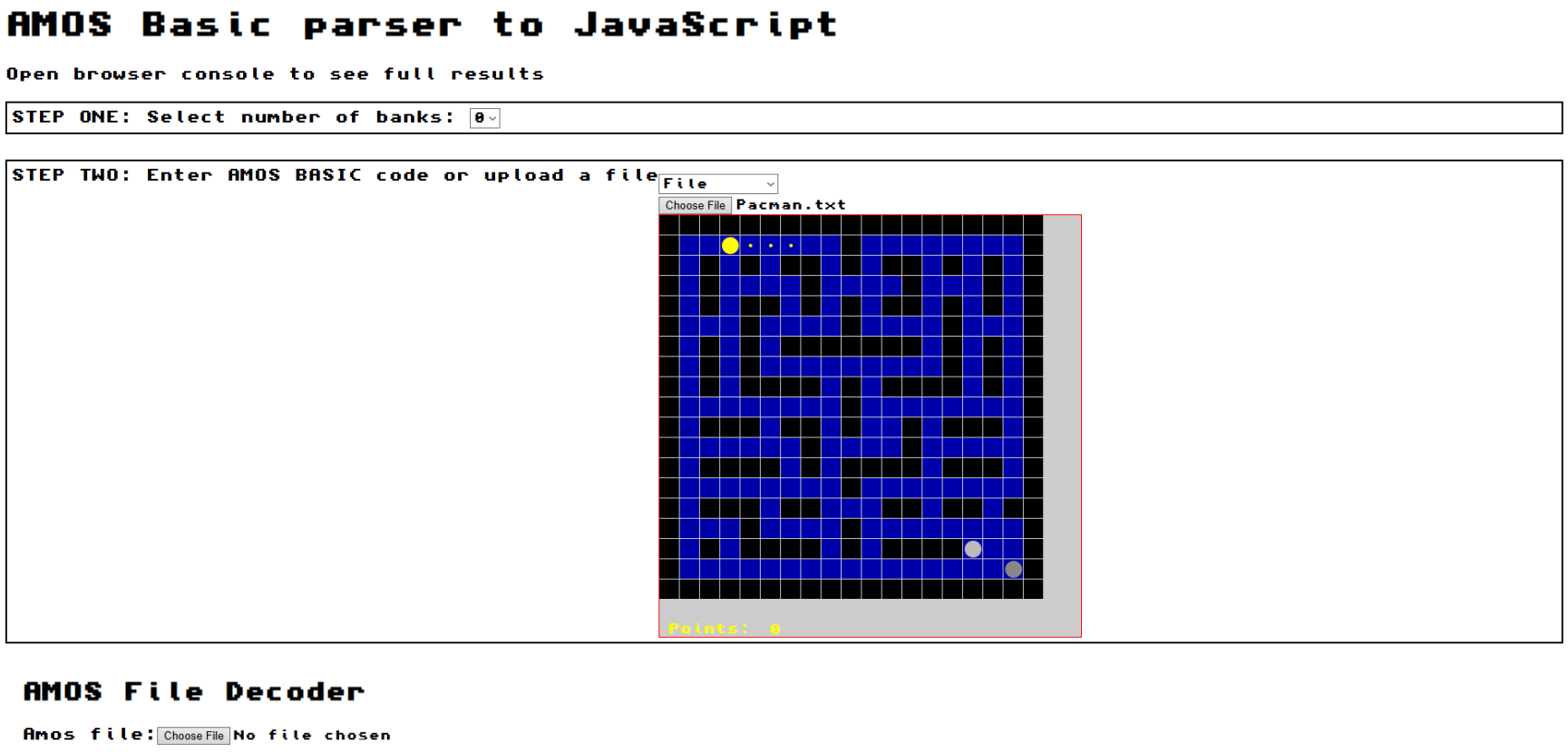
Current Progress

- Procedures
  - Follow the file format
  - **BUT** encrypted ones
  - Decrypt using some simple (?) C code

```
/* read 16-bit big-endian word from unsigned char[] */
#define amos_deek(a) ((((a)[0])<<8)|((a)[1]))
/* read 32-bit big-endian word from unsigned char[] */
#define amos_leek(a) ((((a)[0])<<24)|(((a)[1])<<16)|(((a)[2])<<8)|((a)[3]))

void AMOS_decrypt_procedure(unsigned char *src) {
    unsigned char *line, *next, *endline;
    unsigned int key, key2, key3, size;

    /* src should be a pointer to a line with the PROCEDURE token on it */
    if (amos_deek(&src[2]) != 0x0376) return;

    /* do not operate on compiled procedures */
    if (src[10] & 0x10) return;

    size = amos_leek(&src[4]);
    line = next = &src[src[0] * 2]; /* the line after PROCEDURE */
    endline = &src[size + 8 + 6]; /* the start of the line after END PROC */

    /* initialise keys */
    key = (size << 8) | src[11];
    key2 = 1;
    key3 = amos_deek(&src[8]);

    while (line < endline) {
        line = next; next = &line[line[0] * 2];
        for (line += 4; line < next;) {
            *line++ ^= (key >> 8) & 0xFF;
            *line++ ^=  key       & 0xFF;
            key = (key & 0xFFFF0000) | ((key + key2) & 0xFFFF);
            key2 = (key2 + key3) & 0xFFFF;
            key = (key >> 1) | (key << 31); /* rotate right one bit */
        }
    }
    src[10] ^= 0x20; /* toggle "is encrypted" bit */
}
```

Current Progress

- Current UI with a simple PacMan running



Future Work

- Finish decoder to read AMOS files entirely
- Choose and implement more commands
- Deploy CRVJA to be accessed by anyone
- Improve project organization
  - Local execution
  - Documentation
  - Contributions