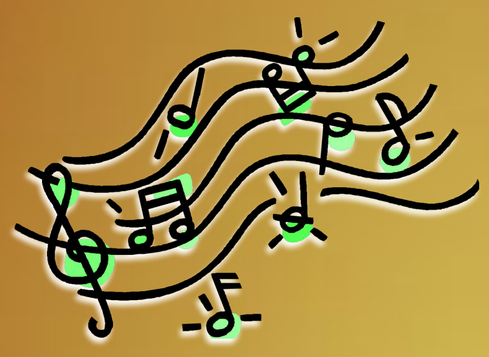


BitTune

Translating your Pixels into a Melody.

Ben Miller, Prakhar Jalan, Tony Wang

April 11, 2019

What is BitTune?

- BitTune is a **GUI application** that allows users to translate their creative drawings into a **simple musical melody**
- Within the app is the **simply defined BitTune language**, that recognizes musical notes and basic shapes within its syntax; which can be **compiled** & **drawn** onto the canvas at any point
- The **BitTune Image file (.bti)** is our **unique file type** created for the purpose of **saving and loading images** drawn/coded onto the canvas

Why is it AMAZING?

- We are trying to seamlessly **fuse auditory art** with **visual art**, and essentially provide a **simplified**, **fun** and **creative** way to produce melodies
- With an **easy-to-use GUI** and **intuitive language**, we encourage the **younger generation** to playfully doodle with BitTune, while also learning valuable **math & logic principles** in relation to programming.

How does it work?

Choose your Approach

You may **DRAW** using the cursor with different color options (notes) and brush sizes OR write **CODE** that can be translated into shapes and drawn onto the canvas whenever needed.

DRAW

- Draw a picture on the blank canvas using the **cursor**
- Select a brush **Colour**, each paired with a musical note (A - G) OR select a **Brush Size** (Thin/Medium/Thick)
- **Clear** the entire canvas at any point

Implementation Explanation

- Canvas is split horizontally into 40 subsections and vertically into 4 subsections, forming a 4 by 40 gridspace
- Maximum of 4 notes can be played simultaneously (4 note chord)
- Unit of time is fixed to quarter notes; thus the maximum song length is 40 quarters, or 10 bars
- Each mouse click/hold events are stored as a coordinate [x1, x2, y1, y2, colour]. An example coordinate would be: [40, 60, 0, 80, 'green']

Load/Save (Optional)

- Load a previously created image onto the canvas at any point
- Once you are satisfied with your drawing, you can choose to save your image

Implementation Explanation

- The 'Save Image' button will store the cursor coordinates into a .bti file
- The 'Load Image' button will read the .bti file and draw the image according to the coordinates

CODE

- Enter code into the text editor using the custom BitTune language

BitTune Language Summary

- Supports **musical notes** A - G
- Uses R to indicate a rest (no sound playing)
- A number in front of the note represents the number of sequential notes

- **Example:** "2ABC 4R" will play notes ABC at the same time for 2 beats and then rest for 4 beats

- Language also supports **drawing shapes** (rect, line, dots, and oval).
- Saving and loading code snippets through .btu (Bitune Language) files

- **Example:** "square(abe) oval(c d d)" will play the chord A-B-E and draw it as a square. The C note followed by D twice, will be drawn as ovals on the canvas

Compile & Draw

- To draw the code, you must click the 'Compile Code to Canvas' button
- This will translate the code into its appropriate shapes, and draw it upon the canvas

Implementation Explanation

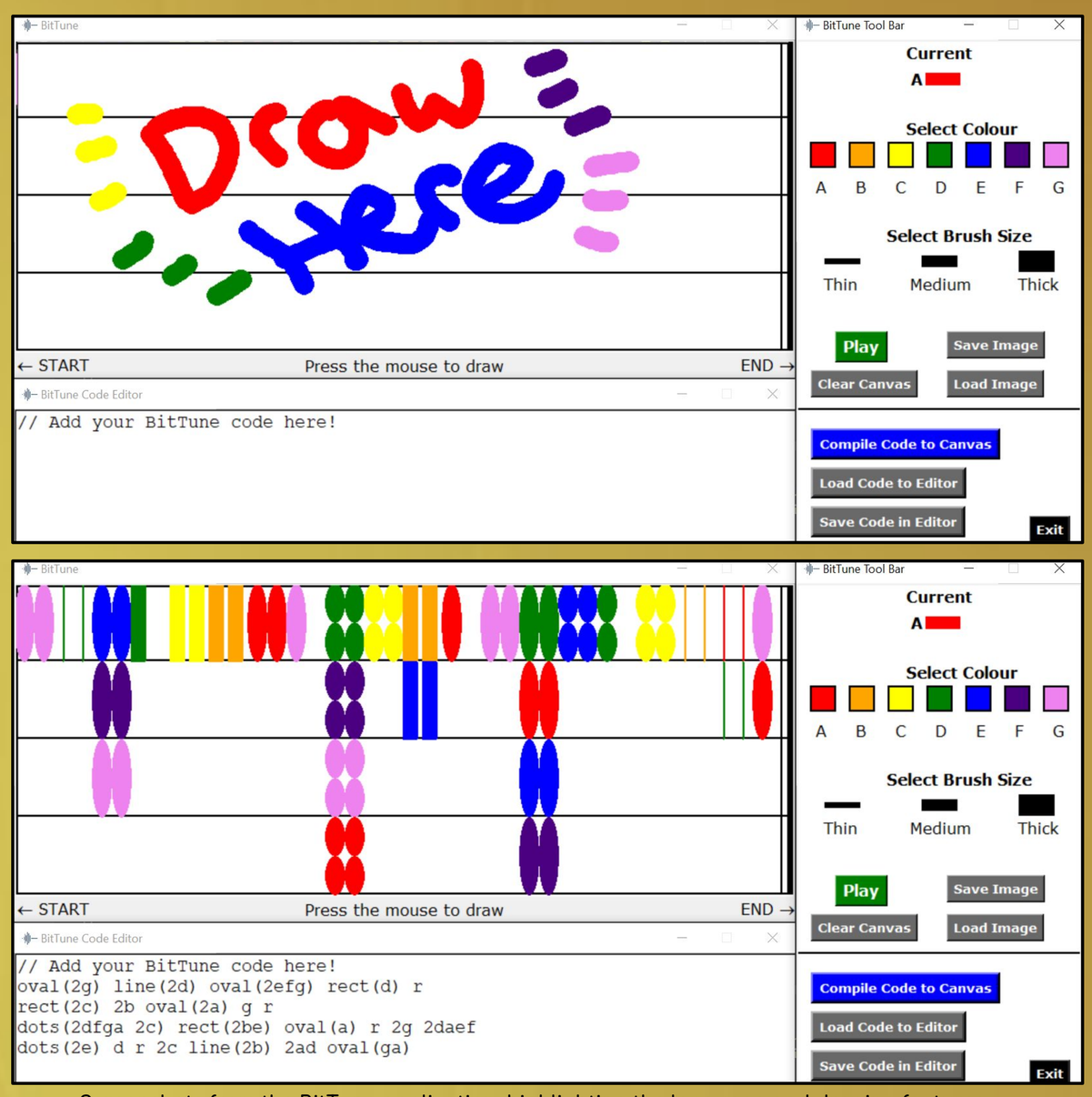
- The text editor will tokenize and parse the input
- The result will be saved as a valid BitTune Image (.bti) file that can be loaded into the a the BitTune Application through the load function

PLAY

Finally, the user can listen to the produced melody by clicking on the 'Play' button!

Implementation Explanation

- The midpoint of each coordinate is computed and mapped to the closest subsection
- Each subsection only stores the most recent color value (note) and empty subsections are treated as rests. Our .wav generator is optimized by removing rests that do not affect the song.
- Our Lexer.py then converts these subsections into a string of notes in the form [Note 1, Note 2, ... NoteN]. Finally the notes are transformed into a tuple that the Python Pysynth Library can convert into 4 separate .wav files (1 for each line) and then overlaid to output the final result.



Screenshots from the BitTune application, highlighting the language and drawing features

```
grammar BitTuneGrammar ;

program: statement+ ;

statement : noteSequence | multiNoteSequence | shapeSequence | commentSequence ;

noteSequence : note ;
multiNoteSequence : number note ;
shapeSequence : shapeName (noteSequence | multiNoteSequence)+ endShape ;

number : INT ;
note : NOTE ;

shapeName : RECT | OVAL | LINE | DOTS ;
endShape : CLOSE_BRACKET ;

commentSequence : COMMENT ;

NOTE : [a-gA-GrR][a-gA-GrR]?[a-gA-GrR]?[a-gA-GrR]? ; // Note can be up to 4 instances of a-g or r for rest

RECT : 'rect(' ;
OVAL : 'oval(' ;
LINE : 'line(' ;
DOTS : 'dots(' ;

CLOSE_BRACKET : ')' ;

COMMENT : '//' ~[\r\n]* '\r'? '\n' -> skip ;

INT : [0-9]+ ;

WS : [ \t\r\n]+ -> skip ; // skip tabs, spaces, and new lines
```

Grammar for the BitTune language

Development Statistics

- Number of Lines of Code: **~2,000 lines**
- Size of documentation: **~400 lines**
- Number of test cases: **15**

Insight Gained

" Adding more elements such as drum loops or advanced orchestration attempts to bring this children's app to a more technical and customizable level; a baby step towards achieving a simplistic Digital Audio Workstation. " - *The BitTune Team 2019*

Implementation

How was it tested?

- Unit testing and black box testing.
- Had a variety of test songs and language files that ensured the different components of our application were working as designed

How is it documented?

- Python Docstring for every function and class

Challenges

- Deciding how many subsections to divide the grid in order to produce moderately elegant sounding music
- Merging the .wav files together to play 4 notes simultaneously in a reasonable time frame. Needed to use multiprocessing to achieve this
- Constructing the ANTLR grammar and Lexer rules
- Incorporating HCI design elements into the GUI

Related Articles

Wilson, Sarah J. "The Cognitive and Academic Benefits of Music to Children: Facts and Fiction." Taylor & Francis, Informa UK Limited, 20 Aug. 2006, http://bit.ly/2VRETBK

WKafai, Yasmin B., and Quinn Burke. "Computational Participation: Teaching Kids to Create and Connect Through Code." SpringerLink, Springer, Cham, 1 Jan. 1970, http://bit.ly/2XfC1Pt