Final Presentation
Dynamic Music Composition in Video Games

Tomo Buchberg, Ahrin Meguerian, Austin Lucky, Keith Choi, Robert Loya, Tristen Khatibi

Department of Computer Science
California State University, Northridge

Outline

- Introduction
- Background
- Related Work
- Goals
- Design - Game/Music
- Implementation - Characters/State Machines/Puzzles/Sequencers
- Sound design and music controllers
- Demo of Project
- Conclusion and future work
- Acknowledgement
- References

Introduction

- Importance of the topic
  - Procedurally generated audio and overall player experience
- Gaps or lack of research
  - Standard of music in the gaming industry
  - The effects of procedurally generated audio in video games
  - Procedurally generated audio and the mood of the player
- Problems solved
  - How effective procedurally generated audio can be in a 2D game
  - Enhance player experience in video games

Player attacking ballbot enemy

Introduction

- How did we solve the problem?
    - By creating a 2D video game using the GODOT engine
    - Integrating various audio plugins
    - By introducing a sequencer to mix game play events and compose music
    - Using an event manager
        - To manage which game events will influence game stages

GODOT
Game engine

Background Knowledge

- Music composition/performance software:
  - Sequencers and Trackers
  - Synthesizers and Drum machines
- Audio data signal processing and real-time audio synthesis
- Godot game engine
- Pure Data programming language

Related Work - Evolution of Music Using Machines

- Pre Written/Composed By Humans
  - The most traditional form of video game music
  - Machine are used to create the sounds rather than the music
- Synthesized by machines
  - Adaptive vs linear music structures

Related Work - Machine Made Video Game Music - Neural Networks

- CNN (Convolutional Neural Network)
    - Use to provide audio sound effects or specific samples
- ANN (Artificial Neural Network)
    - Model is composed of different neurons/connections
    - Not rules based

Related Work - Machine Made Video Game Music - Procedural Generation (PG)

- Music Based on Multiple Algorithms and Parameters
- Experience Based PG
    - Creating Music that evolves in real time
- Adaptive Audio
    - Music whose volume, tune and rhythm changes depending on what is happening in the game

Related Work - Machine Made Video Game Music - Real Time Audio Synthesis

- Processing Power
    - How much is to much?
    - What is the optimal amount of power for sound retrieving and producing?
- Physics and Perception
    - Base off different collisions in a game
    - Mode Compression and Quality Scaling

Goals

- Group Goals:
  - This is a Video Game project so on the Video Game side the goal is to produce a game that runs well and is fun to play.
  - On the Music side we aim to create different conditions that produces dynamic music that is also good to listen to.
- Approach:
  - Game Side: Using GODOT engine as a backbone, 2D retro style Sci-fi game.
  - Music Side: Implement a sequencer using GODOT's audio stream.
    - Finite State Machine to give sequencer input on what and how to play music.

Design

Two major components:

- Game Engine
- Music Engine

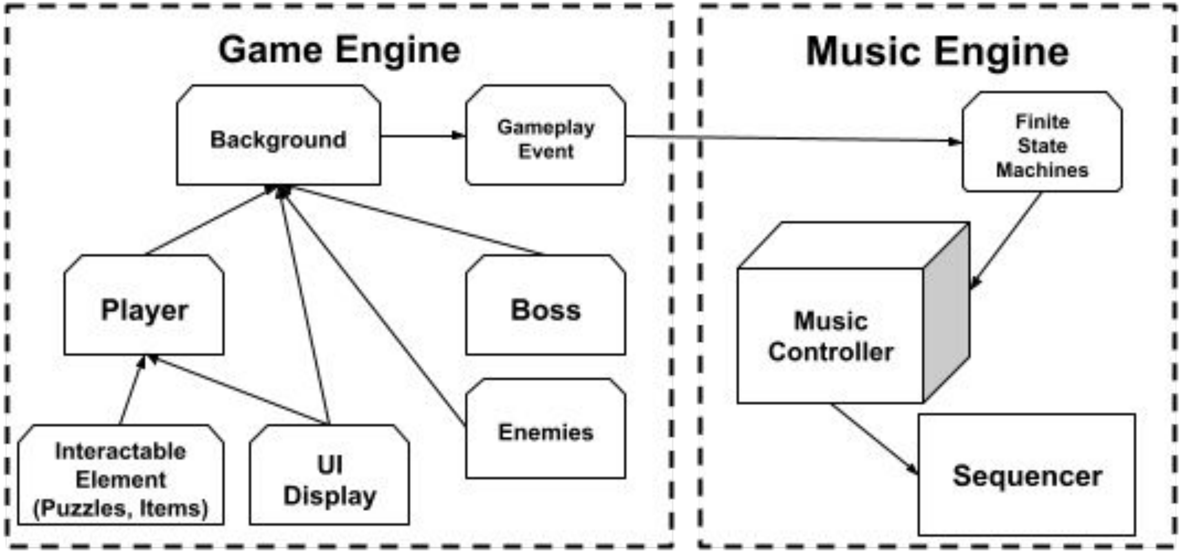
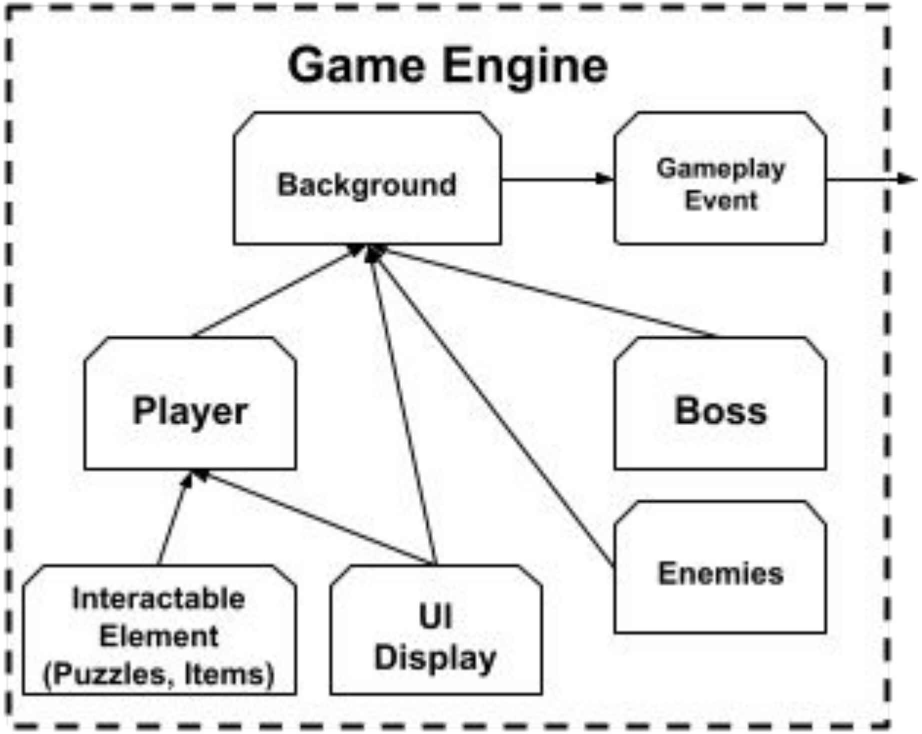


Fig. 1. Framework for the project's design

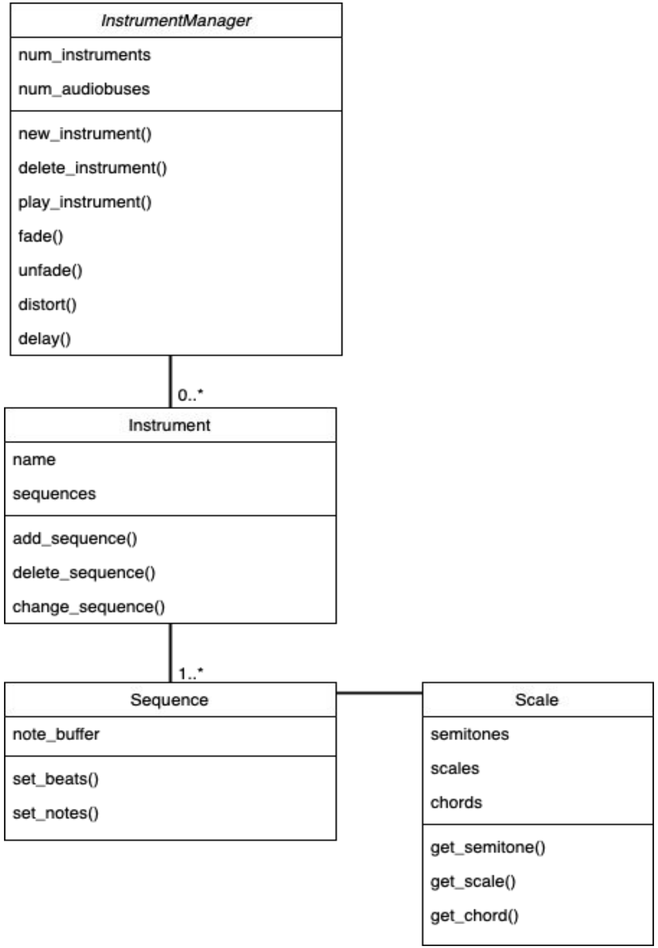
Game Engine

- Player
  - Interactable Elements
- Enemy
- Boss
- UI Display (GUI)
- Gameplay events sent to Music Engine
  - Enemy detected
  - Health lost

Fig. 2. Game Engine design

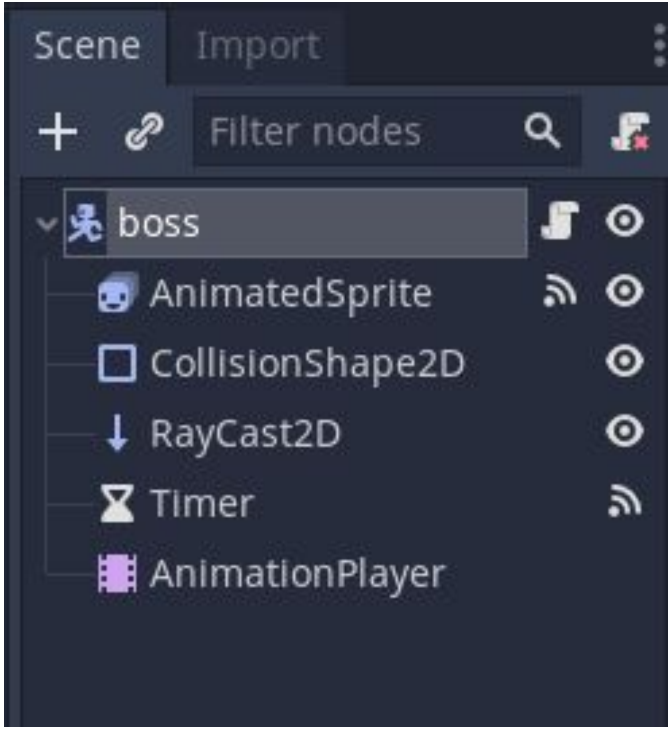


Music Engine



Characters

- Player
  - Keyboard Input Controlled
- Enemies
  - Automatic
- Boss
  - Shoots fireballs



Puzzles

- Sliding
- Switches

State Machines

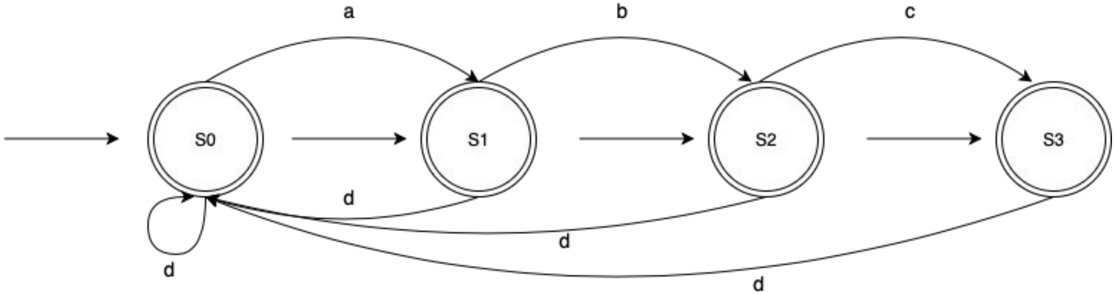
- State Machines implemented within the Godot Engine
  - Control some aspects of the game.
  - E.G. enemy attack states.
- State Machines feed to the music controller
  - Health State Machines take the data of the player’s health and feed to the music controller.

```
if(health<=19 && health>0 && state.name != "19% Health"):
	change_to("19% Health")
elif(health<=49 && health>19 && state.name != "49% Health"):
	change_to("49% Health")
elif(health<=79 && health>49 && state.name != "79% Health"):
	change_to("79% Health")
elif(health>79 && state.name != "100% Health"):
	change_to("100% Health")
```

```
func enter():
	if DEBUG:
		print("79% Health")
	emit_signal("health_1")
```

```
func _on_79_Health_health_1():
	print("Health: 50 ~ 79.")
	setPitch(DEFAULT_PITCH + 1)
	sendMessage()
```

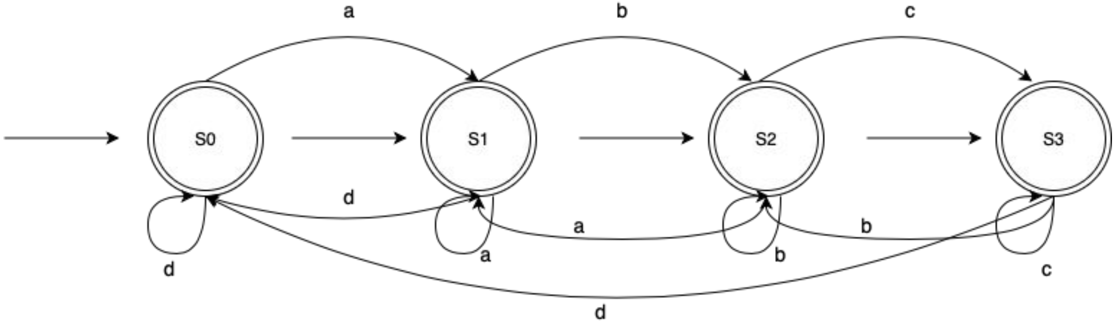
Major Game Events



States
S0: **Start Game**.
S1: **Checkpoint 1**.
S2: **Checkpoint 2**.
S3: **Level finished**.

Input
a: **Puzzle solved.** (Player obtains boss room key).
b: **Boss room entered.** (Player fights boss).
c: **Boss defeated**. (Level is finished).
d: **Restart level.**

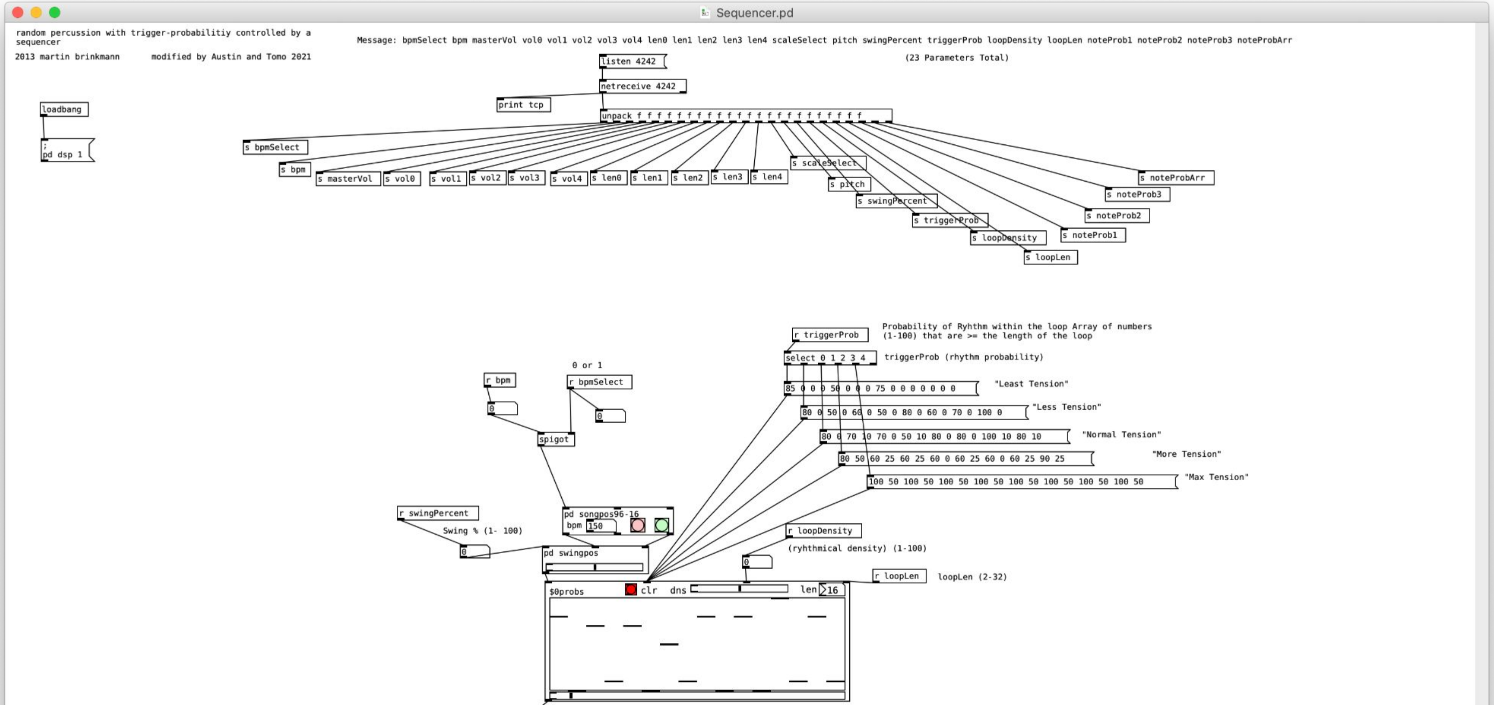
Minor Game Events



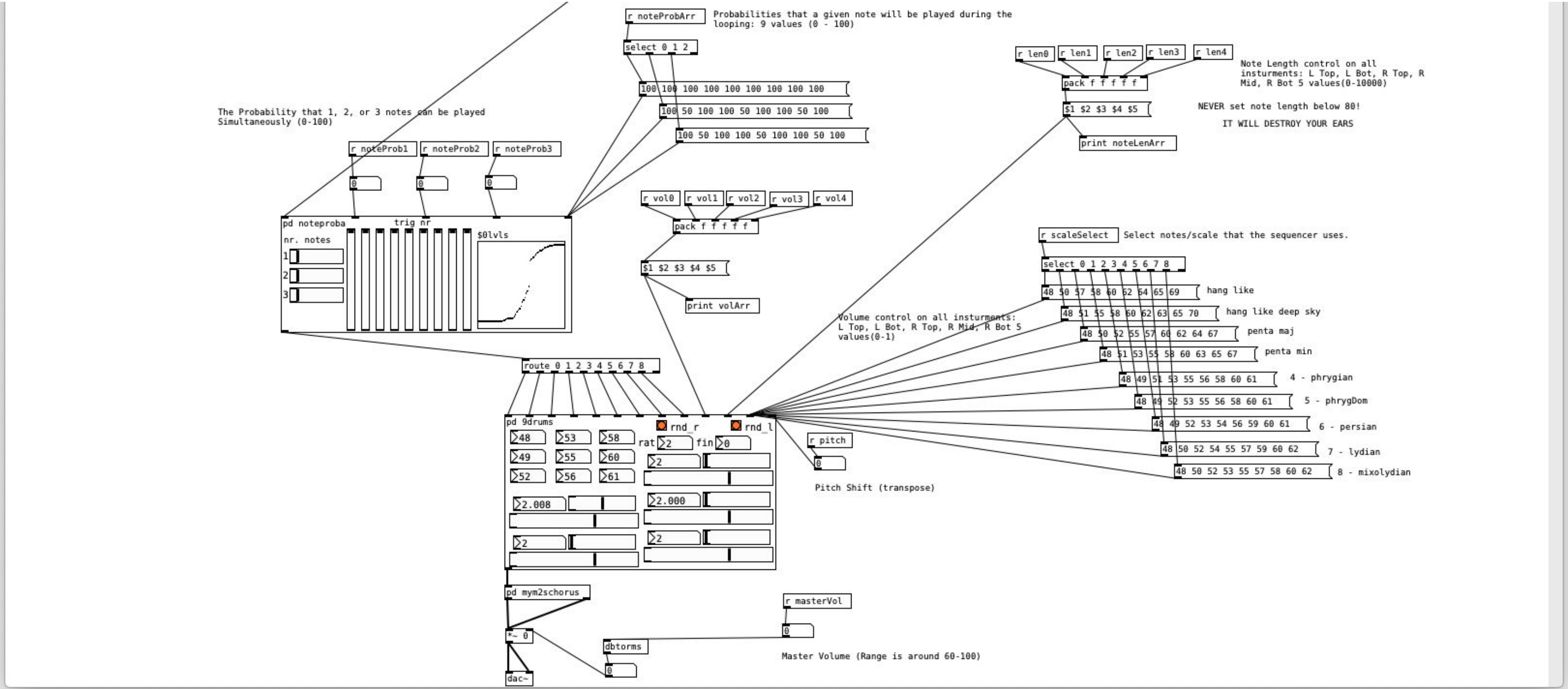
States
S0: **Calm.** (Far away from enemies).
S1: **Induce tension**. (Enemies nearby).
S2: **Build tension.** (Enemies are alerted).
S3: **Most excitement.** (Fight enemies).

Input
a: **Enemies nearby.**
b: **Enemies alerted.**
c: **Battle enemy.**
d: **No Enemies nearby.**

Sequencer

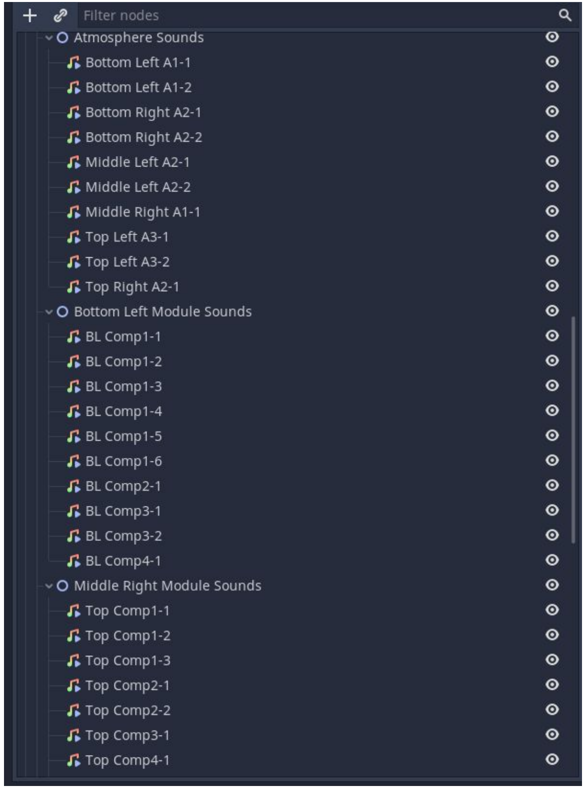


Sequencer.pd
random percussion with trigger-probabilitiy controlled by a sequencer
2013 martin brinkmann
modified by Austin and Tomo 2021
Message: bpmSelect bpm masterVol vol0 vol1 vol2 vol3 vol4 len0 len1 len2 len3 len4 scaleSelect pitch swingPercent triggerProb loopDensity loopLen noteProb1 noteProb2 noteProb3 noteProbArr
(23 Parameters Total)
listen 4242
netreceive 4242
print tcp
loadbang
; pd dsp 1
unpack f f f f f f f f f f f f f f f f f f f f f f f
s bpmSelect
s bpm
s masterVol
s vol0
s vol1
s vol2
s vol3
s vol4
s len0
s len1
s len2
s len3
s len4
s scaleSelect
s pitch
s swingPercent
s triggerProb
s loopDensity
s loopLen
s noteProb1
s noteProb2
s noteProb3
s noteProbArr
r triggerProb
Probability of Ryhthm within the loop Array of numbers (1-100) that are >= the length of the loop
select 0 1 2 3 4
triggerProb (rhythm probability)
85 0 0 0 50 0 0 0 75 0 0 0 0 0 0 0
"Least Tension"
80 0 50 0 60 0 50 0 80 0 60 0 70 0 100 0
"Less Tension"
80 0 70 10 70 0 50 10 80 0 80 0 100 10 80 10
"Normal Tension"
80 50 60 25 60 25 60 0 60 25 60 0 60 25 90 25
"More Tension"
100 50 100 50 100 50 100 50 100 50 100 50 100 50 100 50
"Max Tension"
r bpm
0 or 1
r bpmSelect
spigot
r swingPercent
Swing % (1- 100)
pd songpos96-16
bpm 150
pd swingpos
r loopDensity
(ryhthmical density) (1-100)
r loopLen
loopLen (2-32)
$0probs
clr
dns
len 16


r noteProbArr
Probabilities that a given note will be played during the looping: 9 values (0 - 100)
select 0 1 2
100 100 100 100 100 100 100 100 100
100 50 100 100 50 100 100 50 100
100 50 100 100 50 100 100 50 100
The Probability that 1, 2, or 3 notes can be played Simultaneously (0-100)
r noteProb1
r noteProb2
r noteProb3
pd noteproba
trig nr
nr. notes
$0lvls
r len0
r len1
r len2
r len3
r len4
pack f f f f f
$1 $2 $3 $4 $5
print noteLenArr
Note Length control on all insturments: L Top, L Bot, R Top, R Mid, R Bot 5 values(0-10000)
NEVER set note length below 80!
IT WILL DESTROY YOUR EARS
r vol0
r vol1
r vol2
r vol3
r vol4
pack f f f f f
$1 $2 $3 $4 $5
print volArr
Volume control on all insturments: L Top, L Bot, R Top, R Mid, R Bot 5 values(0-1)
r scaleSelect
Select notes/scale that the sequencer uses.
select 0 1 2 3 4 5 6 7 8
48 50 57 58 60 62 64 65 69
hang like
48 51 55 58 60 62 63 65 70
hang like deep sky
48 50 52 55 57 60 62 64 67
penta maj
48 51 53 55 58 60 63 65 67
penta min
48 49 51 53 55 56 58 60 61
4 - phrygian
48 49 52 53 55 56 58 60 61
5 - phrygDom
48 49 52 53 54 56 59 60 61
6 - persian
48 50 52 54 55 57 59 60 62
7 - lydian
48 50 52 53 55 57 58 60 62
8 - mixolydian
route 0 1 2 3 4 5 6 7 8
pd 9drums
rnd_r
rnd_l
rat
fin
r pitch
Pitch Shift (transpose)
pd mym2schorus
*~ 0
dbtorms
dac~
r masterVol
Master Volume (Range is around 60-100)

Sound Design

- Sequencer is not the only sound source needed for an immersive gaming experience
- Perfectly Timed and Mixed Sound Effects



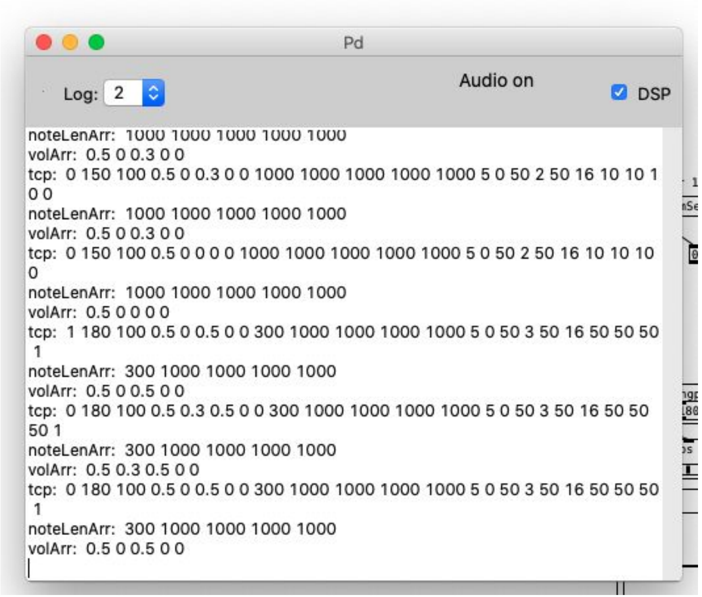
Music Controller

```
func _ready():
	# Connect to Pure Data
	socket = StreamPeerTCP.new()
	print("connecting to Pure Data...")
	socket.connect_to_host("127.0.0.1", 4242)
	sendMessage()
```

The Music Controller is a script in our game responsible for changing the music's state and updating the sequencer patch.

- Establishes a local TCP socket for sending packets to be received by the patch via Pure Data's FUDI protocol.

- Everytime the music's state changes, a message is sent to update the sequencer.

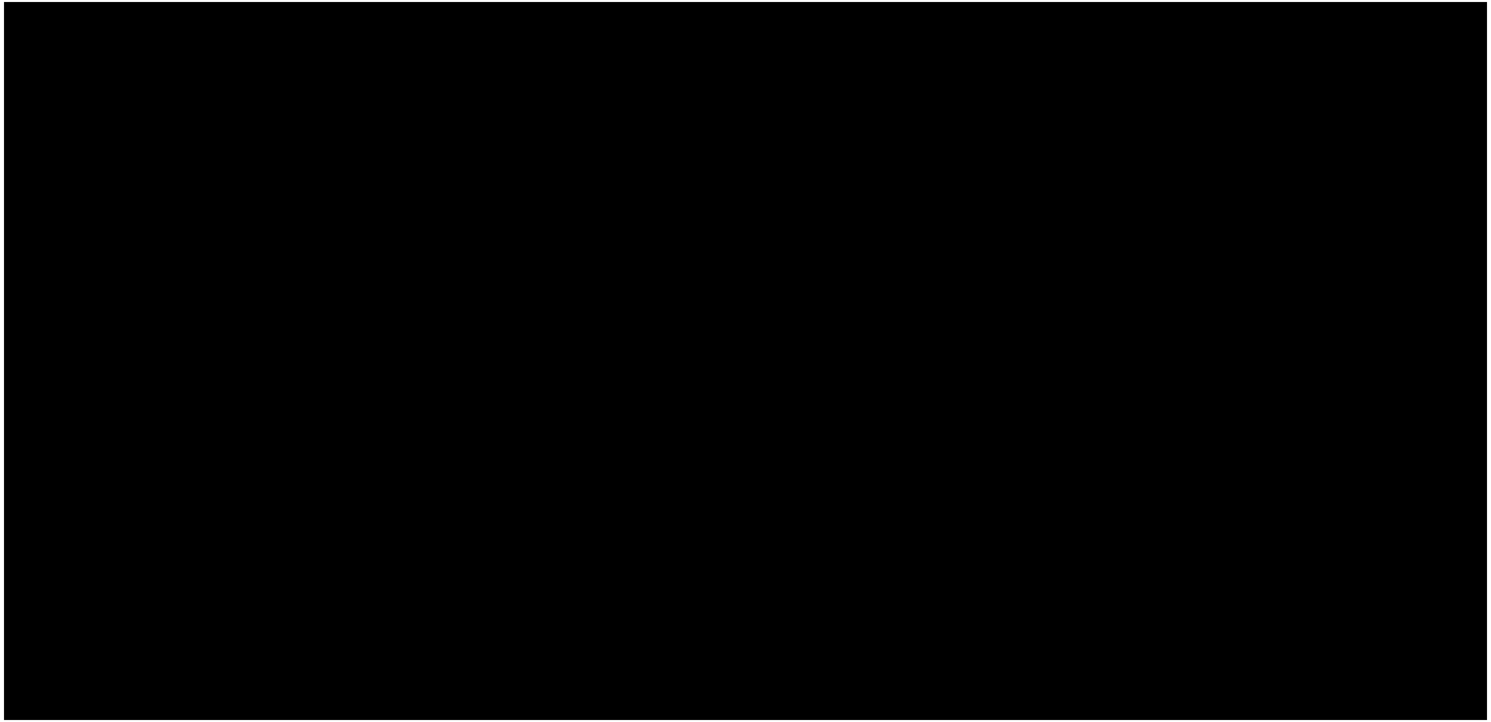


Music Controller (cont.)

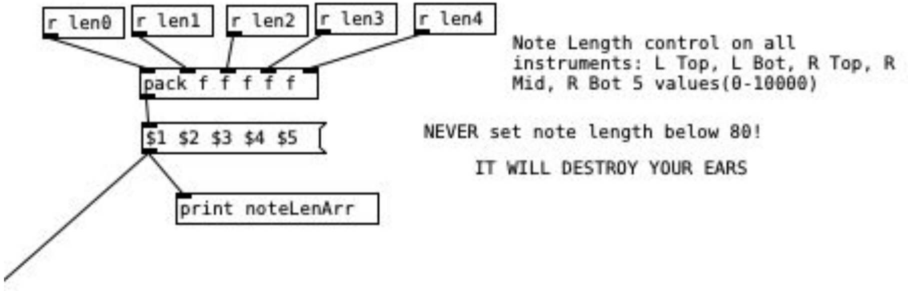
```
## Areas
func _on_Area1_body_entered(body):
	in_area1 = true
	if body != null:
		if body.name == "player":
			print("Area1 entered.")
			if area1_visited == false:
				area1_visited = true
				
			setBPMSelect(1) # IMPORTANT: Only change BPM for areas/menus (otherwise "stuttering" occurs often).
			setBPM(150)
			setMasterVol(MAX_MASTER_VOLUME)
			setInstrumentVolumes([FULL, SILENT, SILENT, SILENT, SILENT])
			setInstrumentNoteLengths([NoteLength.LONGEST, NoteLength.LONGEST, NoteLength.LONGEST, NoteLength.LONGEST, NoteLength.LONGEST])
			setScale(Scales.PHRYG_DOM)
			setPitch(DEFAULT_PITCH)
			setSwingPercent(MEDIAN_VALUE)
			setTriggerProb(TriggerProb.NORMAL_TENSION)
			setLoopDensity(MEDIAN_VALUE)
			setLoopLen(DEFAULT_LOOPLENGTH)
			setNoteProb(10, 10, 10)
			setNoteProbArr(LOW)
			sendMessage()
			
			setBPMSelect(0) # IMPORTANT: Remember to turn off BPM change
```

- There are a total 23 parameters which can be modified.
- The Music Controller receives signals from on-screen events and changes to the player’s state, setting new values in response.

Game Demo



Results



- After experimenting, more parameters were added to the music controller.

- Overall, we accomplished our goal of creating a game which composes music dynamically.

- Our game has the ability to procedurally change the music’s:
  - Rhythmic patterns and rhythm probabilities
  - Note selection, note probabilities, and transposition
  - Master volume and BPM
  - Instruments’ note lengths and volumes

Conclusion & Future Work

- Game Development
  - Taught ourselves to work with a given engine, using the documentation and guides.
  - Individually or as teams we can continue development and make the game bigger and prettier.
- Music Development
  - Likewise, the music sequencer can be improved upon, allowing for even more variability.
  - Learned how to take inputs from one program and feed to another
  - Communication between programs can be a vital tool in program development

References

Cachia, W., Aquilina, L., Martinez, H. P., & Yannakakis, G. N. (2014). Procedural Generation of Music-Guided Weapons. 2–3.

Collins, K. (2009). An introduction to procedural music in video games. Contemporary Music Review, 28(1), 5–15. https://doi.org/10.1080/07494460802663983

Duarte, A. E. L. (2020). Algorithmic interactive music generation in videogames. SoundEffects-An Interdisciplinary Journal of Sound and Sound Experience, 9(1), 38–59.

Gaina, R. D., & Stephenson, M. (2019). “did you hear that?” Learning to play video games from audio cues. IEEE Conference on Computatonal Intelligence and Games, CIG, 2019-Augus. https://doi.org/10.1109/CIG.2019.8848088

Khatchatourov, A., Pachet, F., & Rowe, V. (2016). Action identity in style simulation systems: Do players consider machine-generated music as of their own style? Frontiers in Psychology, 7(MAY), 1–9. https://doi.org/10.3389/fpsyg.2016.00474

Koons, N., & Haungs, M. (2019). Intrinsically musical game worlds: Abstract music generation as a result of gameplay. ACM International Conference Proceeding Series, 1–4. https://doi.org/10.1145/3337722.3341833

Lidy, T., & Schindler, A. (2016). Parallel Convolutional Neural Networks for Music Genre and Mood Classification. Music Information Retrieval Evaluation Exchange (MIREX 2016), February, 1–4. http://www.ifs.tuwien.ac.at/~schindler/pubs/MIREX2016.pdf

References

McDonagh, A., Lemley, J., Cassidy, R., & Corcoran, P. (2018). Synthesizing Game Audio Using Deep Neural Networks. 2018 IEEE Games, Entertainment, Media Conference, GEM 2018, 312–315. https://doi.org/10.1109/GEM.2018.8516448

Plans, D., & Morelli, D. (2012). Experience-driven procedural music generation for games. IEEE Transactions on Computational Intelligence and AI in Games, 4(3), 192–198. https://doi.org/10.1109/TCIAIG.2012.2212899

Plut, C., & Pasquier, P. (2020). Generative music in video games: State of the art, challenges, and prospects. Entertainment Computing, 33(December 2019), 100337. https://doi.org/10.1016/j.entcom.2019.100337

Plut, C., & Pasquier, P. (2019). Music matters: An empirical study on the effects of adaptive music on experienced and perceived player affect. IEEE Conference on Computatonal Intelligence and Games, CIG, 2019-Augus. https://doi.org/10.1109/CIG.2019.8847951

Raghuvanshi, N., Lauterbach, C., Chandak, A., Manocha, D., & Lin, M. C. (2007). Real-time sound synthesis and propagation for games. Communications of the ACM, 50(7), 66–73. https://doi.org/10.1145/1272516.1272541

Risi, S., & Togelius, J. (2020). Increasing generality in machine learning through procedural content generation. Nature Machine Intelligence, 2(8), 428–436. https://doi.org/10.1038/s42256-020-0208-z

Q & A

?