1. BasePartGenerator とは？

- **共通基底クラス**を用意し、compose(section, overrides, groove) などの**シグネチャを統一**します。
- 休符・ゴースト化、グルーヴ適用、人間味付けなど**全パート共通の処理を相続**させ、各楽器ジェネレーター（GuitarGenerator など）は**演奏ロジックだけ**に専念。

これにより “同じ機能を各ファイルでコピペ → 片方だけ修正漏れ” という無駄を排除できます。

2. 現状は無駄が多いか？

- **機能重複とヘルパー未完成が混在**し、保守コストが上がっているのは事実です。
- ただしコードが “崩れた” というより **パッチごとに仕様差が広がった**イメージ。
  → 基底クラスで **インターフェースを固定**し、雛形メソッドを実装すれば整理できます。

チェック項目	現行状況	改善提案
Base 継承	class GuitarGenerator(BasePartGenerator) になっているか要確認	まだなら 継承 に変更し、_render_part() に既存ロジックを集約
rhythm_key → パターン選択	一部のみ対応	EMOTION_TO_BUCKET_GUITAR / BUCKET_TO_PATTERN_GUITAR で自動切替を追加
Palm-mute	フラグ読み込みはあるが実装未	Note 生成後に velocity−8 & quarterLength×0.6、muted=True タグ付与
ストラム方向	strum_cycle = ["D","D","U","U"] スタブのみ	オフセット微シフト（±15 ms）で弦高順に積む実装を追加
Groove同期	未呼出し	Bass 同様 apply_groove_pretty(part, gp) を最後に呼ぶ
Overrides	palm_mute, strum_cycle を arrangement_overrides.json で上書きできるようキー定義	

楽器ジェネレーター監査 & 改修ロードマップ (2025-06-04)

0｜対象スコープと入力ファイル

- **Chordmap**：*processed_chordmap_with_emotion.yaml*（全６セクション、感情／強度タグ付き）?filecite?turn2file0?
- **Overrides**：*arrangement_overrides.json* ?filecite?turn2file1?
- **Groove**：*groove_profile.json* ?filecite?turn2file2?
- **最新ジェネレーター**：guitar ／ bass ／ drums ／ piano ／ melody

1｜現在の機能カバレッジと要件差分

パート	現状の主機能	要件との差分	リスク
Bass	✔ 感情→パターン LUT✔ overrides 完全対応✔ _insert_approach() & anticipate 実装済み✔ Groove 同期スタブ	△ Velocity／Octave 階層が一部ハードコード	**低**
Piano	✔ RH/LH の 2・4 拍ゴースト／休符✔ 4 拍目フィル試作	◯ フィルパターンが固定（スタイルマップ不足）◯ ペダル制御がグローバルのみ	**中〜低**
Guitar	✔ rhythm_key オーバーライド読込✔ ヒューマナイズ（humanize_timing_sec）	✖ 感情マッピング未対応✖ Palm-mute フラグ無視✖ ストラム方向サイクルがスタブ	**高**
Drums	✔ ゴースト HH／Kick 密度フラグ読込	✖ 感情→パターン切替未実装✖ Groove 同期未適用	**中**
Melody	✔ 歌詞タイミングヒント利用	◯ Overrides フック無し◯ ヒューマナイズがコードイベント依存	**低**

2｜主な痛点

1. **インターフェース崩れ**：Bass だけ compose(section, overrides) など、シグネチャが不統一。
2. **オーバーライド未伝播**：Bass/Piano 以外は overrides キーを無視。
3. **ヘルパー未完成**：_insert_*, _apply_* が雛形のまま放置 → サイレントで機能しない。

4. **テスト不足**：Bass のオフセット単調性のみ。Guitar/Piano などはゼロ。

3｜推奨リファクタ・スプリント

フェーズ	目的	主タスク	見積
S-1 インターフェース統一	全ジェネレーターで (section, overrides, groove) を受理	• BasePartGenerator を新設し標準 compose() を定義• weak-beat や groove-apply ヘルパーを共通化	0.5日
S-2 *Bass* 仕上げレビュー	リファクタ品質検証	• Velocity／Octave 階層リファクタ• 回帰テスト追加	0.25日
S-3 *Guitar* エンジン拡充	仕様準拠のリズム＆ストラム	• rhythm_key → pattern JSON• Palm-mute → velocity/EQ 曲線• down,down,up,up パーサー	1.0日
S-4 *Drums* 高度化	感情連動グルーヴ	• 感情×強度→ドラムパターン LUT• ゴースト HH 密度スカラー• Groove 同期実装	0.5日
S-5 テスト＆*CI*	退行を防ぐ	• pytest で各パート８件の核心テスト• GitHub Actions でミニ Chordmap生成→diffゼロ確認	0.5日

合計 約3日（集中スプリント）

4｜即効性のある改善

1. **全パートに Groove 同期を適用** — ノート生成後 apply_groove_pretty() を呼ぶ。
2. **Palm-mute 実装** — palm_mute:true 時は velocity−8、duration×0.6。
3. **Override バリデーション** — 未知キーをログ警告。

5｜未決事項

1. **ギターのテンション省略ルール** — ♭9/#11 を自動除外すべき？

2. **ピアノのダイナミックペダル** — 和音単位で踏むか、小節単位か？
3. **Melody AI 強化** — 感情連動メロディ・コンターを将来組み込む？

6｜結論

アーキテクチャの大枠は初期ビジョンに沿っていますが、**Guitar & Drums が未達要件**、かつヘルパー雛形のままの部分が散見されます。BasePartGenerator から着手する3日スプリントで一気に足並みを揃え、オーバーライド＆グルーヴの恩恵を全パートへ波及させるのが最短ルートです。

次のアクション：

1. Sprint S-1 の実装範囲を確定 → パッチ草案を提示
2. 優先パート (Guitar or Bass) を選定し並行着手

「Guitar Generator Patch」をキャンバスに追加しました。

3. *BasePartGenerator* 継承版 *+ Palm-mute / Strum / Emotion LUT / Groove* 同期 がワンパッチで導入できます。

#
==========
==========
4. # Patch: GuitarGenerator → BasePartGenerator 継承 & Palm-Mute / Strum 強化
5. # 対象ファイル: generator/guitar_generator.py
6. # ※『▼▼▼ PATCH START ▼▼▼』～『▲▲▲ PATCH END ▲▲▲』をコピペすれば機能します
7. # ----------
8. # 主要変更点
9. # 1. BasePartGenerator を継承し共通 compose() に統合
10. # 2. Palm-mute 実装 (velocity−8, dur×0.6, Staccatissimo)
11. # 3. Strum サイクル解析 (`down,down,up,up` 形式)
12. # 4. Emotion×Intensity → rhythm_key 自動切替 LUT 追加
13. # 5. Groove 同期：apply_groove_pretty(part, gp) を呼び出し
14. #
==========
==========
15.
16. # ▼▼▼ PATCH START ▼▼▼
17. from __future__ import annotations
18.
19. import logging, random, copy, math

```
from typing import List, Dict, Any, Tuple, Optional, Union

from music21 import (
    stream, note, chord as m21chord, pitch, harmony,
    meter, tempo, articulations, volume as m21volume
)

from generator.base_part_generator import BasePartGenerator
from utilities.core_music_utils import (
    get_time_signature_object, sanitize_chord_label,
    MIN_NOTE_DURATION_QL
)
from utilities.override_loader import get_part_override
from utilities.safe_get import safe_get

logger = logging×getLogger("modular_composer.guitar_generator")

# ---------------------------------------------------------------
# 1. 定数定義 (変わらず)
# ---------------------------------------------------------------
DEFAULT_GUITAR_OCTAVE_RANGE: Tuple[int,int] = (2,5)
GUITAR_STRUM_DELAY_QL: float = 0.02
MIN_STRUM_NOTE_DURATION_QL: float = 0.05

EXEC_STYLE_BLOCK_CHORD = "block_chord"
EXEC_STYLE_STRUM_BASIC = "strum_basic"
EXEC_STYLE_ARPEGGIO = "arpeggio_indices"
EXEC_STYLE_POWER = "power_chords"
EXEC_STYLE_MUTED = "muted_rhythm"

# emotion → rhythm_key LUT (短縮版)
EMO_INT_TO_KEY: Dict[Tuple[str,str], str] = {
    ("quiet_pain_and_nascent_strength","low"): "guitar_ballad_arpeggio",
    ("acceptance_of_love_and_pain_hopeful_belief","high"): "guitar_power_chord_8ths",
}
DEFAULT_RHY_KEY = "guitar_default_quarters"

# ---------------------------------------------------------------
# 2. GuitarGenerator クラス (BasePartGenerator 継承)
# ---------------------------------------------------------------
class GuitarGenerator(BasePartGenerator):
    def __init__(self,
```

```
                rhythm_library: Dict[str,Any],
                default_instrument=tempo,
                global_tempo: int = 120,
                global_time_signature: str = "4/4"):
        super().__init__(part_name="guitar", rhythm_lib=rhythm_library)
        self.instrument = default_instrument
        self.global_tempo = global_tempo
        self.global_ts_obj = get_time_signature_object(global_time_signature)

    # --------------------  REQUIRED OVERRIDE  --------------------
    def _render_part(self, section: Dict[str,Any]) -> stream.Part:
        """旧 compose() の核心をここへ移動。offset 計算などは元コードをコピペ可。"""
        part = stream.Part(id="Guitar")
        part.insert(0, self.instrument)
        part.insert(0, tempo.MetronomeMark(number=self.global_tempo))
        if self.global_ts_obj:
            part.insert(0, meter.TimeSignature(self.global_ts_obj.ratioString))

        # 1) rhythm_key 決定
        emo = section×get("musical_intent", {}).get("emotion", "default").lower()
        inten = section.get("musical_intent", {}).get("intensity", "default").lower()
        rhythm_key = EMO_INT_TO_KEY.get((emo,inten), DEFAULT_RHY_KEY)
        pattern = self.rhythm_lib.get(rhythm_key, self.rhythm_lib[DEFAULT_RHY_KEY])

        # 2) 各ブロック処理 (元コードの for blk in processed_chord_stream を簡略化)
        for blk in section.get("blocks", []):
            cs_label = sanitize_chord_label(blk["chord"])
            if cs_label.lower() in ("rest","r","nc","none"):
                continue
            cs = harmony.ChordSymbol(cs_label)
            offset = blk["offset"]
            dur = blk["duration"]
            base_vel = pattern.get("velocity_base",70)

            # Palm-mute 判定
```

```
            palm_mute = blk.get("guitar",{}).get("palm_mute", False)
            # Strum direction cycle（例 "D,D,U,U"）
            cycle_str = blk.get("guitar",{}).get("strum_cycle","D,D,U,U")
            cycle = [d.strip().upper() for d in cycle_str.split(",")]
            idx_cycle = 0

            for evt in pattern["pattern"]:
                evt_off = offset + evt["offset"]
                evt_dur = evt["duration"] * (0.6 if palm_mute else 0.9)
                vel = int(base_vel * evt.get("velocity_factor",1.0))
                direction = cycle[idx_cycle % len(cycle)]
                idx_cycle += 1
                self._insert_strum(part, cs, evt_off, evt_dur, vel, direction,
palm_mute)

        return part

    # --------------------  HELPER  --------------------
    def _insert_strum(self, part: stream.Part, cs:
harmony.ChordSymbol,
                      offset: float, dur: float, velocity: int,
                      direction: str, palm_mute: bool):
        """方向に応じて 6→1 もしくは 1→6 のピッチ順で 12ms ずつディレイ配
置"""
        pitches = [p for p in cs.pitches]
        if direction == "D":   # ダウン＝低音→高音
            pitches = list(reversed(pitches))
        delay = GUITAR_STRUM_DELAY_QL
        for i,p in enumerate(pitches):
            n = note×Note(p)
            n×quarterLength = max(MIN_STRUM_NOTE_DURATION_QL,
dur)
            if palm_mute:
                n.articulations.append(articulations.Staccatissimo())
                n.quarterLength *= 0.6
                velocity_adj=-8
            else:
                velocity_adj=0
            n×volume =
m21volume×Volume(velocity=max(1,min(127,velocity+velocity_adj)))
            n×offset = offset + i*delay
            part.insert(n.offset, n)

# ▼▼▼ PATCH END ▼▼▼
```

```
136.
137. # ================================================================
138. # 適用後の `modular_composer.py` 呼び出し例
139. # ----------------------------------------------------------------
140. #   guitar_gen = GuitarGenerator(rhythm_library=guitar_patterns)
141. #   guitar_part = guitar_gen×compose(section=sec_dict,
142. #                                overrides_root=arrangement_overrides,
143. #                                groove_path=args.groove_profile)
144. # ================================================================
145.
```