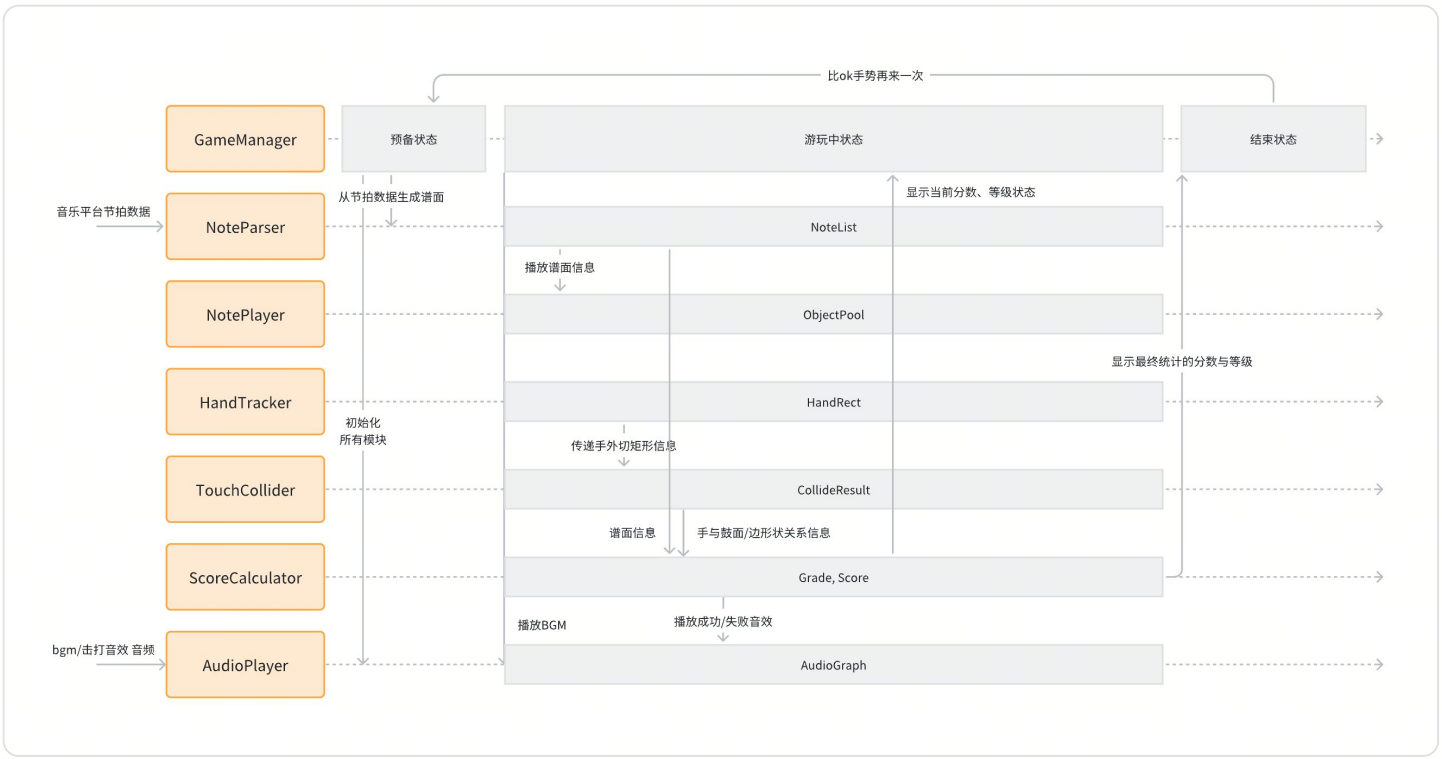
互动特效音乐类游戏研发要点总结：太鼓达人为例

模块一览



1. 状态机管理

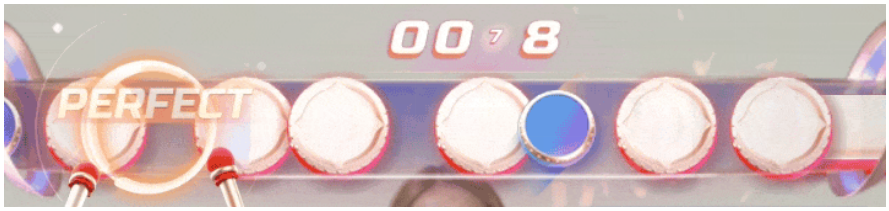
- 分为【预备状态】【游玩中状态】【结束状态】三种状态，控制在不同状态展现不同的游戏界面，播放不同的画面元素，结束状态下比OK能够重置所有游戏信息、回到预备状态重新开始

2. 节奏谱面生成

太鼓达人游戏规则是：上方指示区飘过“鼓面”“鼓边”“鼓面连击”“鼓边连击”四种音符类型，玩家需要根据上方指示，在音符到达CheckPoint的瞬间击打鼓面或鼓边，根据击打时机的准确度来评分。

“鼓面”音符

“鼓边”音符



谱面播放过程

谱面指的是这些音符随着音乐节奏规律出现的顺序：音乐类游戏中，谱面是否节奏合理、难度适中，占玩家体验比重很大。

2.1 指定内置歌曲的情况

在指定内置播放音乐为《一笑江湖》的情况下，需要预先用json文件记录每个音符出现的时间点与音符种类。

由于一首歌的谱面包含上百个音符，手动一一指定json内数据内容，工作量非常大且不直观；

故我们另起了一个专门用于预先人工打谱的工程，根据触屏时间，点击屏幕左半边记为"O"（鼓面），点击右半边记为"X"（鼓边），保持长按记为连击"-"；播放结束后，工程记录每次点击屏幕的时间点以及连击音符的时长，转为json文件保存到本地。

```
onEvent(_event) {
  if (_event.type === Amaz.EventType.TOUCH) {
    const touch = _event.args.get(0);
    const noteType = touch.x < 0.5 ? "O" : "X";
    if (touch.type == Amaz.TouchType.TOUCH_BEGAN) {
      this.touchBeginTime = this.timeStamp;
      this.preNoteType = noteType;
    } else if (touch.type == Amaz.TouchType.TOUCH_ENDED) {
      const duration = this.timeStamp - this.touchBeginTime;
      if (duration > 1000 && noteType == this.preNoteType) {
        this.touchQueue.push({
          time: this.touchBeginTime,
          type: noteType + "-",
          duration: duration,
        });
        console.log("add note" + noteType + "-");
      } else {
        this.touchQueue.push({
          time: this.touchBeginTime,
          type: this.preNoteType,
        });
        console.log("add note" + noteType);
      }
    }
```

点击事件生成谱面音符记录

```
    {
        "time": 5825.000047683716,
        "type": "O"
    },
    {
        "time": 6426.000118255615,
        "type": "O"
    },
    {
        "time": 6718.000173568726,
        "type": "X"
    },
    {
        "time": 7163.000106811523,
        "type": "O"
    },
    {
        "time": 7631.999969482422,
        "type": "O-",
        "duration": 1370.0001239776611
    },
    {
        "time": 9121.000051498413,
        "type": "O"
    },
    {
        "time": 9567.000150680542,
        "type": "X"
    },
```

最终保存到本地的json样式

2.2 玩家自行在曲库中选择音乐的情况

互动特效支持玩家在曲库中选择自己喜欢的音乐进行游玩，我们不可能对曲库中大量的音乐资源一一人工打谱，所以需要总结出从曲库资源自动转换成谱面的一套规律算法。

按照互动特效接曲库协议，从音乐平台可以拿到一首歌的节拍信息"beats"和副歌（高潮）时间区间信息"chorus"，结构体类似于：

```
{
    "beats": {
        "time": [3010,3760,4500,5260,6040,6780,7560, ...],
        "value": [1,2,3,4,1,2,3,...],
```

```
        "energy":
    [199.95552,21.912996,227.0227,-163.64697,150.54747,-86.56395,-544.66187,...
    ]
    },
    "chorus": {
        "start_time": 145200,
        "duration": 78200
    }
}
```

按照音乐类游戏一般的谱面规律，遵循：1. 节拍越重，音符越多越复杂，游玩难度越高 2. 副歌时间区间内的游玩难度比非副歌时间要高

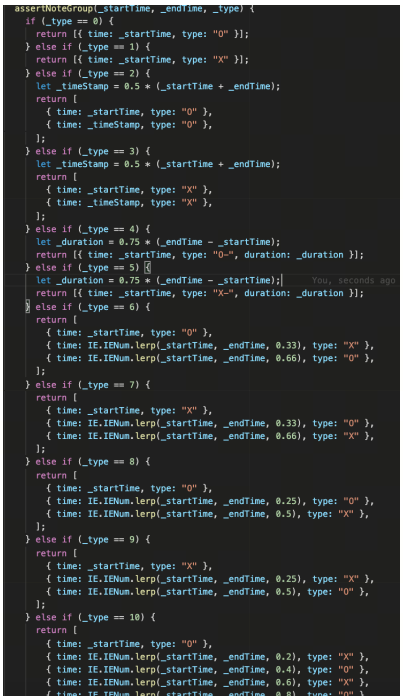
所以我们根据拿到的信息进行处理，类似于：

- 太鼓达人的谱面可以是在节奏切分的基础上，每一拍之内应用几个格式：（O：红，X：蓝，-：长音）
  - O
  - X
  - OO
  - XX
  - O--
  - X--
  - OXO
  - XOX
  - OOX
  - XXO
  - OXOXO
  - XOXOX

（难度大致从上到下递增）

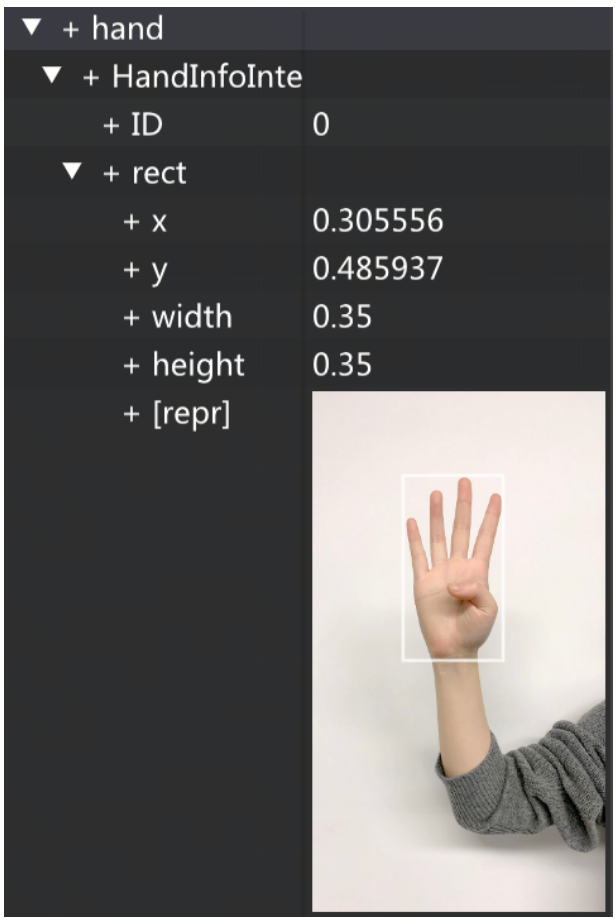
- 取每个小节第一个重拍的时间点（即"value"为1对应的time毫秒数），检查到相应的节拍强度"energy"
- 根据energy所在区间，分配这个小节内的鼓点节拍类型；其中如果落在副歌时间内，自动在原来的节拍类型基础上难度层级加一级
- 将相应的鼓点节拍类型插入到json object里

```
for (let idx = 0; idx < _beats.time.length; idx++) {
  if (_beats.value[idx] === 1) {
    let _type = Math.floor((_beats.energy[idx] + 1100) / 200);
    if (
      _beats.time[idx] > _chorus.start_time &&
      _beats.time[idx] < _chorus.start_time + _chorus.duration
    ) {
      _type += 2;
    }
    let _prevList = this._noteList;
    let _curList = this.assertNoteGroup(
      _beats.time[idx],
      _beats.time[idx + 4],
      _type
    );
    this._noteList = _prevList.concat(_curList);
  }
}
```

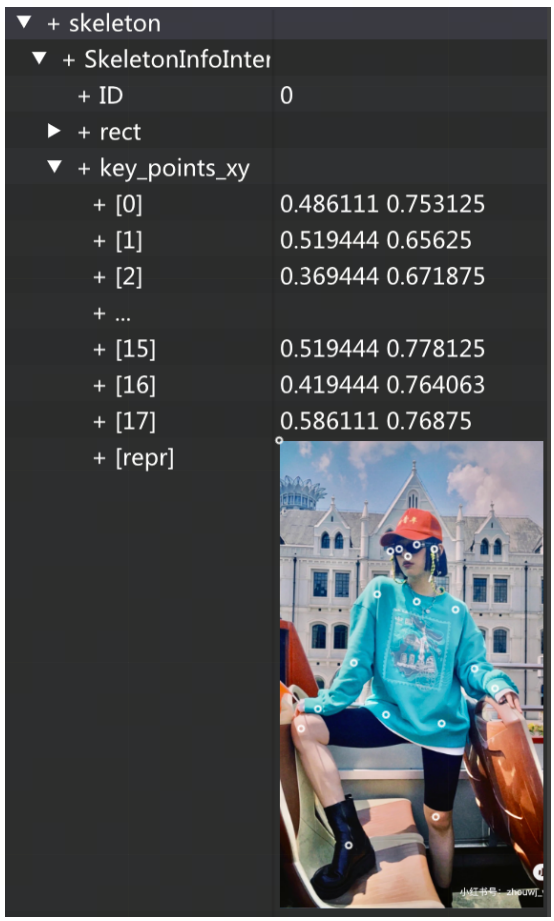


3. 手部跟踪

一般手机音乐类游戏是通过判断玩家触摸屏幕的事件来游玩，我们这里的特殊之处是需要让观众感知到主播游玩过程，所以是通过算法识别主播手部坐标来代替“触摸”“击打”过程。现有BACH算法里比较相关的是hand与skeleton



hand



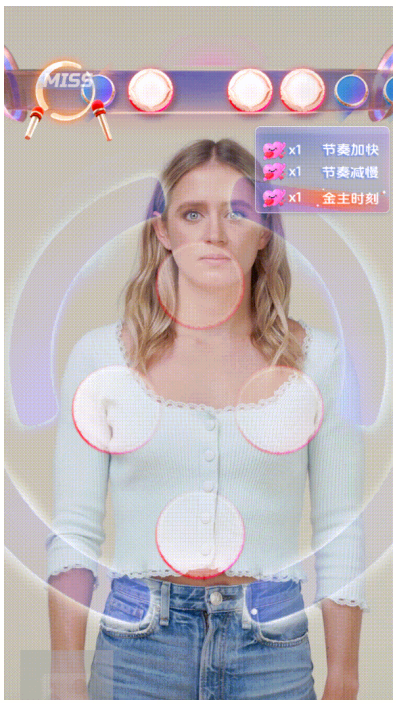
skeleton

3.1 分辨是用hand还是skeleton

- 只有hand算法或只有skeleton算法，就用只存在的那个算法
  - 与人体距离相机远近有关，太近的时候会只有hand，太远只有skeleton
- 两者都有：根据实际测试两种算法较佳的作用范围，当handInfo.scale > 0.5 用hand，否则用skeleton
  - 临界状态大概比这张图更接近屏幕一点，这张图大概是0.4⬇

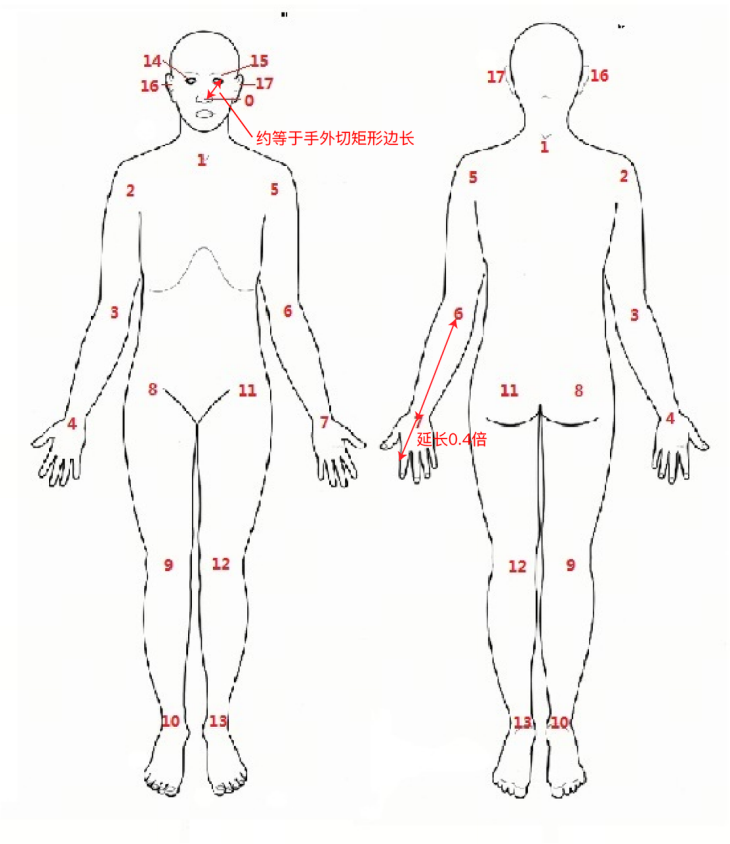
3.2 调用不同算法结果的处理方法

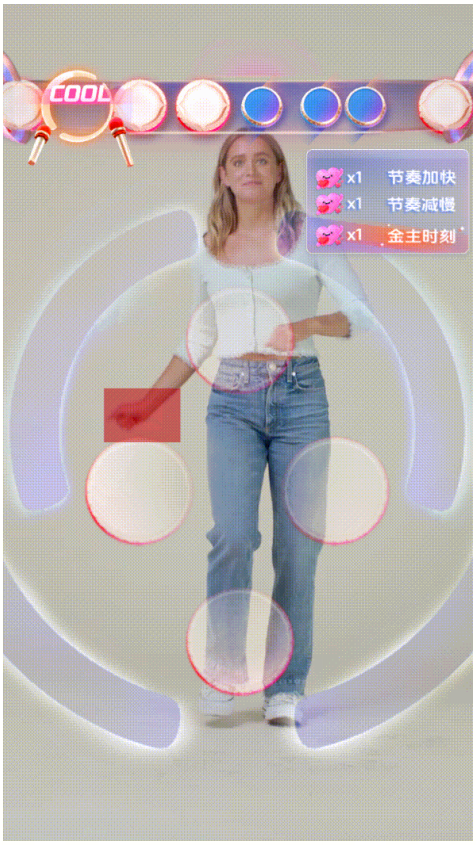
- hand➡取外切rect，判断rect与判定区域的位置关系



- skeleton➡取手腕点位，经过一番计算后模拟手掌外切矩形
  - 矩形大小：取skeleton点位中眼睛与鼻子距离，设为矩形边长。（人体比例绘画知识：一般人手的大小刚好可以参照半张脸。）

- 矩形中点：取手肘到手腕的向量，在手腕位置基础上偏移这一向量*0.4（Magic Number）



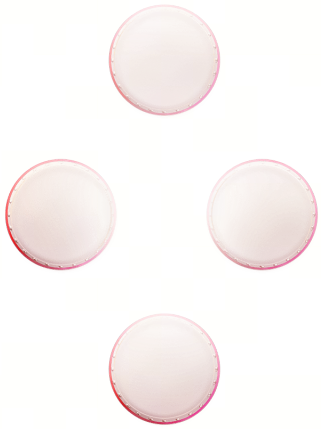


3.3 手保持不动的时候不算击打

- 防止主播太水，手一直摁着鼓面处不动就全算击打了
- 如果手中心点的位置超过三秒的偏移<某个阈值，记为数据无效

4. 识别击打

这是一个难点，尤其因为鼓边的形状是不规则特殊形状，引擎内没有对应形状的Collider可以直接调用，需要自行在脚本里判断人手外切矩形和鼓面、鼓边的坐标位置关系。

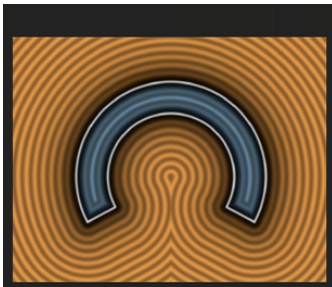


鼓面形状

鼓边形状

方案一 用SDF判断

- 一开始考虑用SDF距离场公式来判断手击打位置是否在这一形状内部



Ring - exact (https://www.shadertoy.com/view/DsccDH)

```
float sdRing( in vec2 p, in vec2 n, in float r, float th )
{
    p.x = abs(p.x);

    p = mat2x2(n.x,n.y,-n.y,n.x)*p;

    return max( abs(length(p)-r)-th*0.5,
                length(vec2(p.x,max(0.0,abs(r-p.y)-th*0.5)))*sign(p.x) );
}
```

放弃原因：

a. 我们并不是判断单点与弧形的位置关系，而是矩形与弧形的相交关系

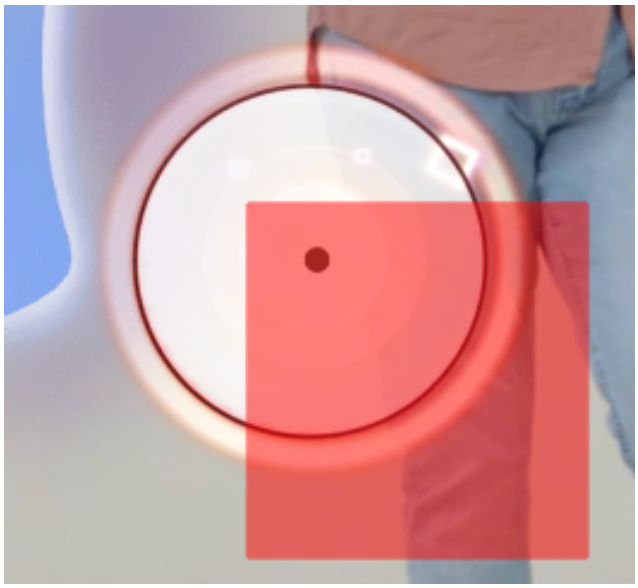
b. 计算每个点具体在距离场中返回的数值，有些浪费，我们只需要判断“是否在范围内”的关系即可

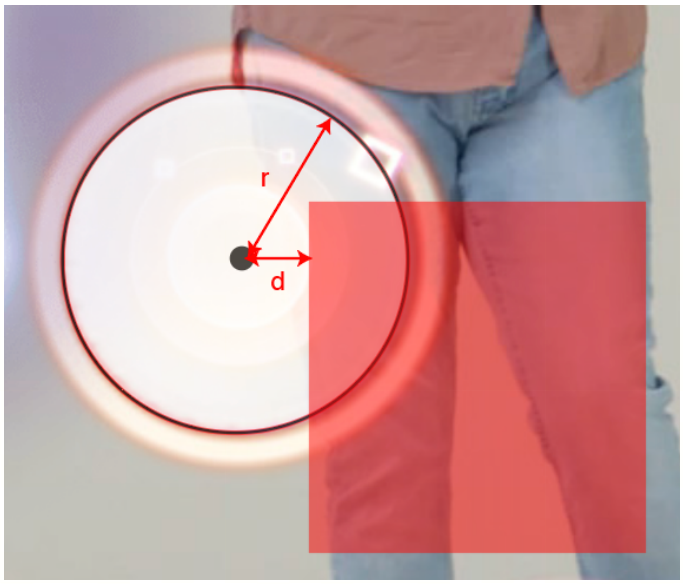
方案二 坐标位置的几何关系判断

- 与鼓面圆：

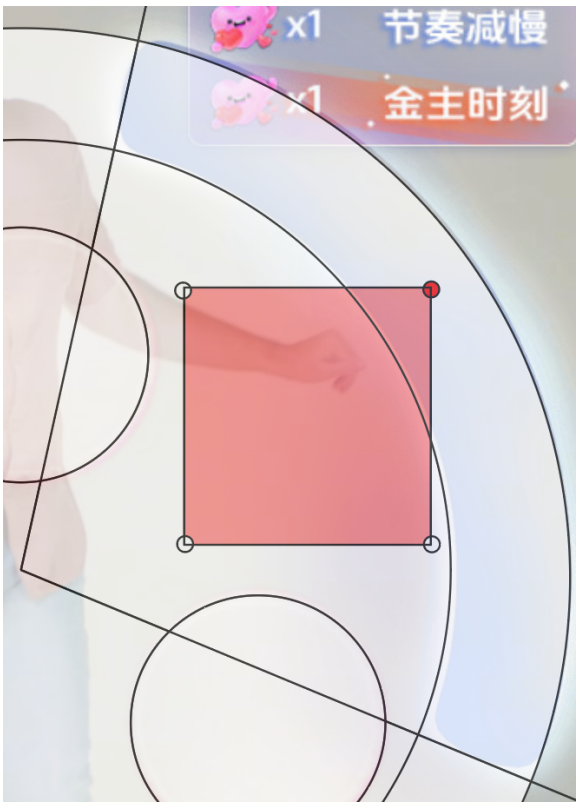
i. 判断鼓面圆心是否在手掌外切矩形之内：当满足这一条件，视为击打到鼓面，return true;

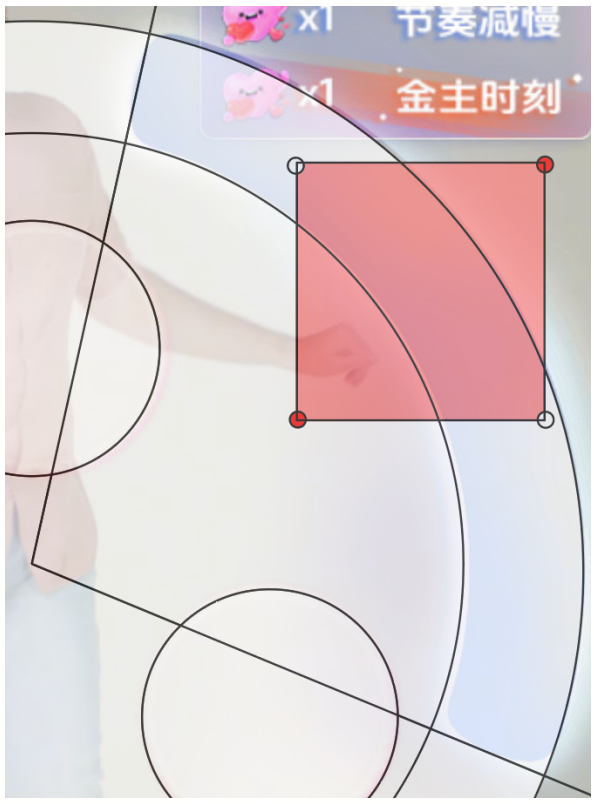
ii. 再判断鼓面圆心到矩形四边的最短距离是否小于半径，如果有一边距离小于鼓面圆半径，视为击打到鼓面。





- 与鼓边弧：
  - 手掌外切rect四点转换到极坐标，其对应的半径、角度，满足下列条件其一即视为交叉
    i. 任意有一个点的半径在内圈、外圈之间；且角度在范围中内
    ii. 有一个点在外圈半径外侧，且另有一个点在内圈半径内侧；两个点都在角度范围内



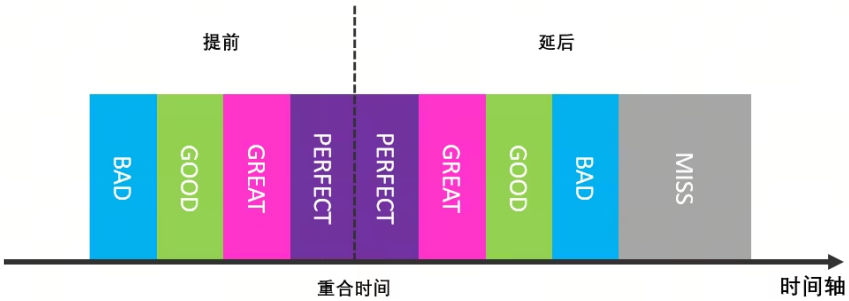


所以是要返回四点的极坐标半径在范围中还是内外；是否在角度范围内；然后统计四点的情况来判断

5. 计算分数

普通音符

参考图

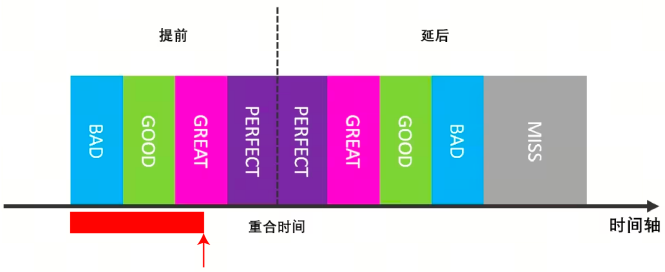


以每个拍子间隔为基准，与最佳的击打时间距离：

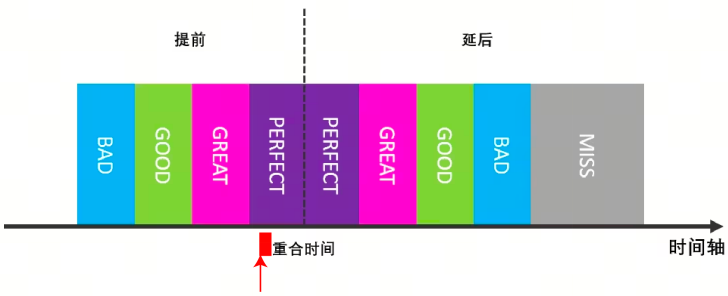
- Perfect：±10%
- Cool：±20%
- Good：±50%
- Miss：>50%

其中：在一个音符的判定时间区域

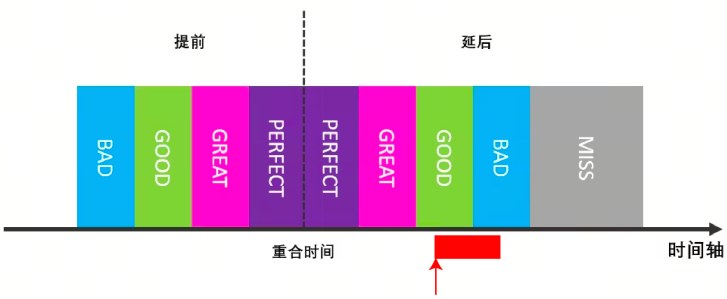
1. 当前处于重合时间之前的区间：当前处于哪个级别范围，就更新临时级别记录到哪个级别；当前到达“perfect”范围，直接返回最终结果为“perfect”，并取消监听这一音符
2. 当前处于重合时间之后的区间：当前处于哪个级别范围，直接返回最终结果为这个范围。
3. 当前处于“miss”区间：之前有关于该音符的临时级别记录，返回最终结果为那一临时记录；如果没有该音符任何临时级别记录，直接返回最终结果为“miss”



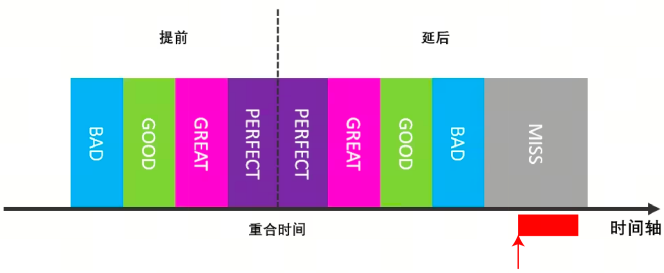
例1:结果为GREAT



例2:结果为PERFECT



例3:结果为GOOD



例4:结果为MISS

长条音符

- 权重50%：开头音符的准确度，参见普通音符；miss了之后，不再对后面继续判定

- 权重50%：在其后长条时间内hold了百分之多少

> - Perfect：>90%
> - Cool：>80%
> - Good：>50%

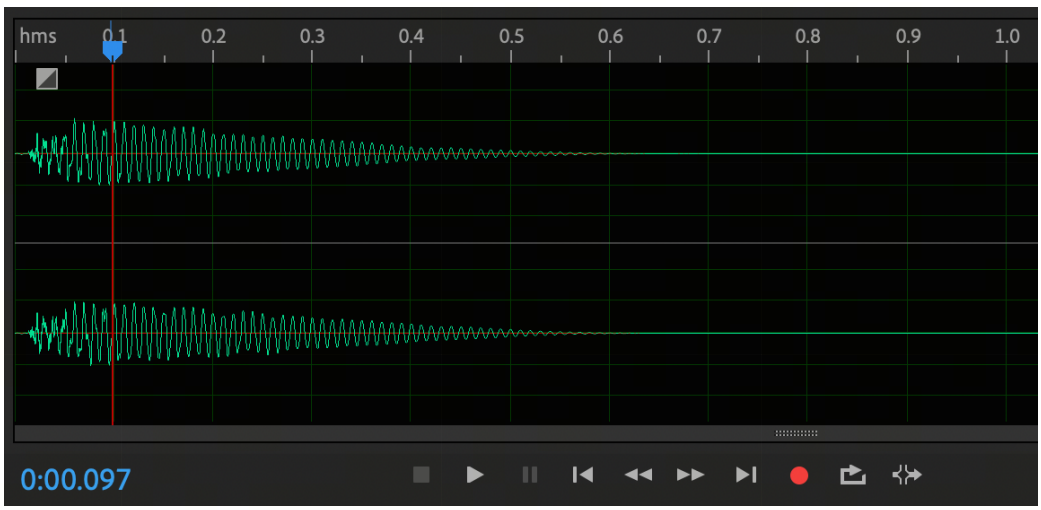
最终结算

以达到总分的百分比计算

> - >80%：节奏感大师
> - 60%~80%：超人气鼓手
> - 30%~60：打击乐萌新
> - 0～30%：拨浪鼓玩家

6. 播放音频

击打到鼓面或鼓边的声音，不能够在击打到的瞬间再播放音频：看波形图，我们耳朵认为是“击打瞬间”的声音，其实是在“开始播放“之后的0.097秒



这种情况需要注意：当音符还有0.097到达CheckPoint，且玩家的手已经处于击打状态，那么就可以提前开始播放击打音效。否则用户感知到的是延迟了0.097秒才有敲击声，产生音画不同步问题