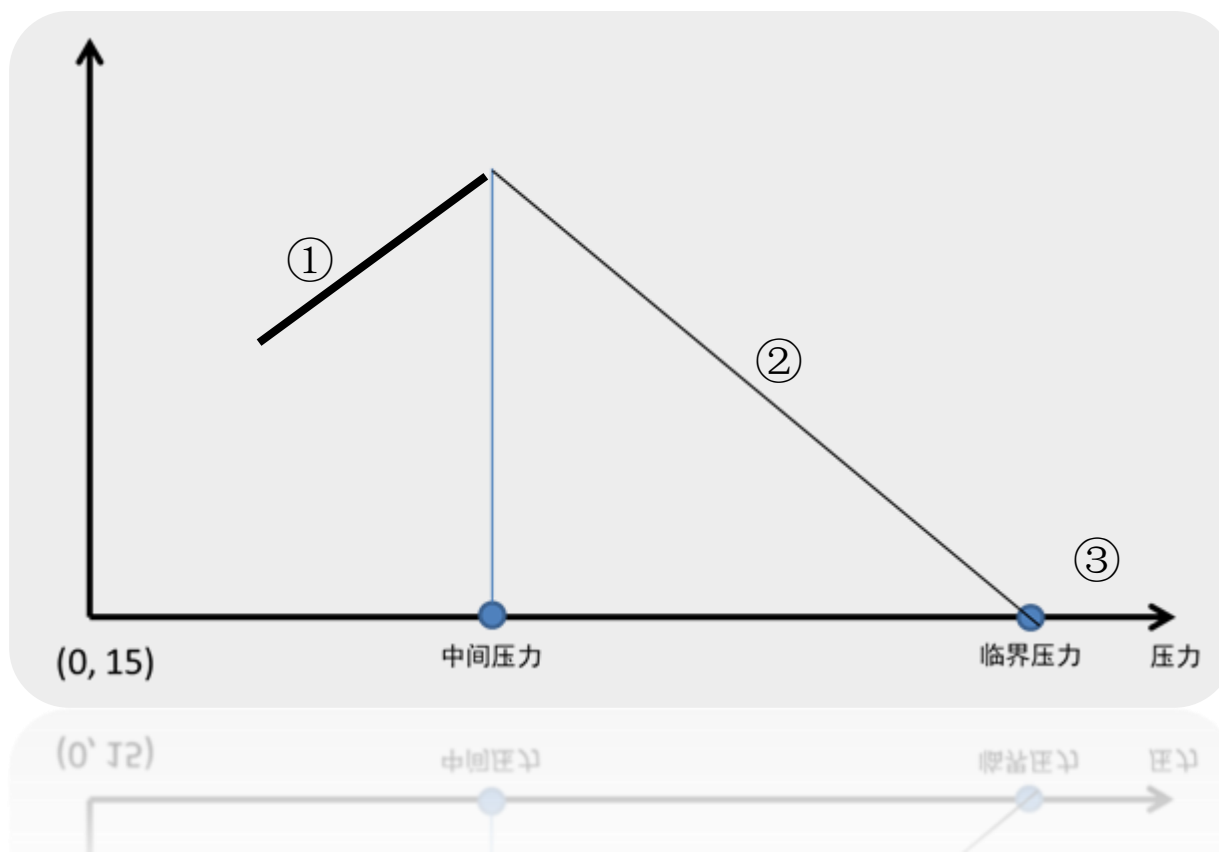
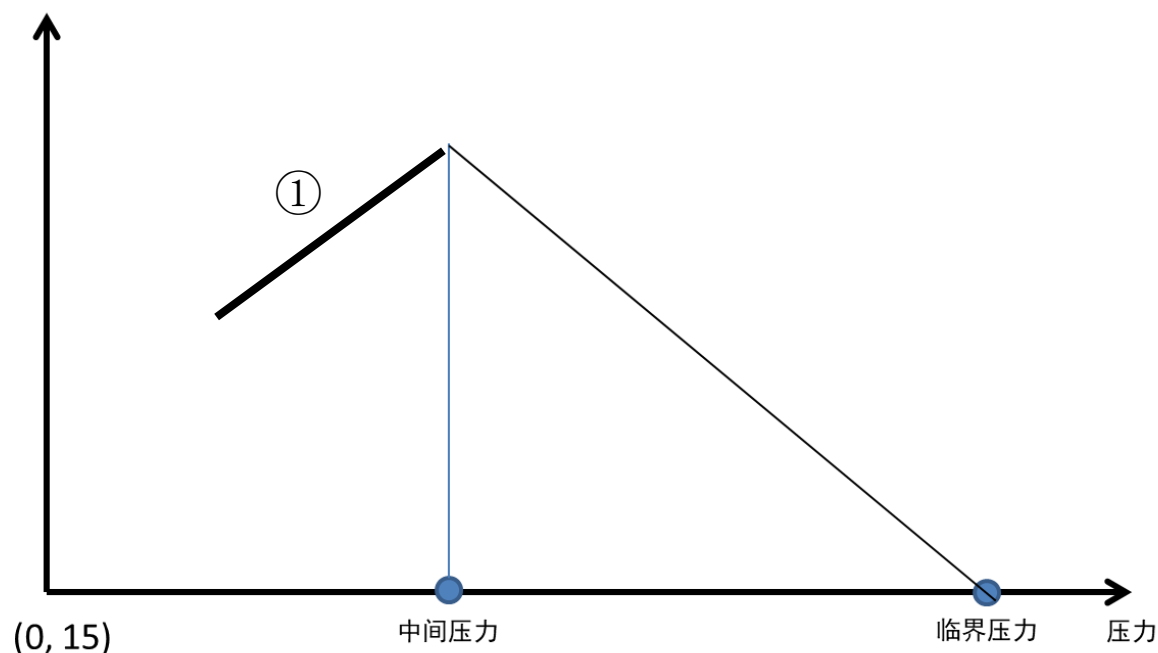


三阶段玩法&逻辑代码实现部分说明



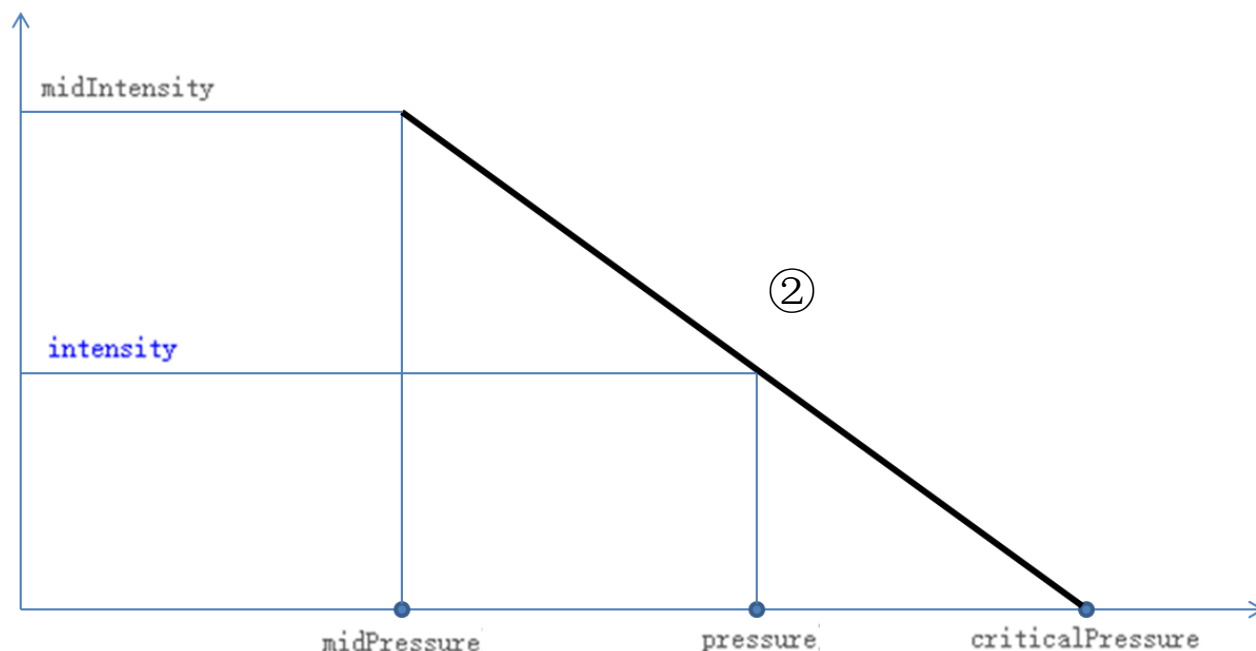
一阶段，压力小于或等于中间压力，强度从零缓慢增加。

```
93     this._runtime.unRandomIntensity = 0;
94     this._runtime.targetIntensity = 0;
256     // 低于中间压力，逐步提升目标强度，应用随机扰动，中间强度实时更新为当前强度
257     else if(pressure <= cfg.midPressure){
258         const increaseIntensity = dtSec * (cfg.intensityGradualIncrease || 0);
259         this._runtime.unRandomIntensity += increaseIntensity;
260         const rnd = 1 + (Math.random() - 0.5) * 2 * (cfg.stimulationRampRandomPercent / 100);
261         this._runtime.targetIntensity = Math.min(Math.max(this._runtime.unRandomIntensity * rnd,
0), cfg.maxMotorIntensity);
262         this._runtime.midIntensity = this._runtime.currentIntensity;
263     }
```



二阶段，压力介于中间压力和临界压力，强度随肛门压力提升而减弱，随肛门压力减弱而提升，强度随当前肛门压力的事实调节的映射方程为图中黑色的斜线。

```
265 // 介于中间压力和临界压力之间，
266 // 以上方if最后抓取的midIntensity作为midPressure的压力，
267 // 将目标强度以当前压力进行以点 (midPressure,midIntensity)和点 (criticalPressure,0)之间连线的线性映射
268 else if (pressure < cfg.criticalPressure) {
269     const intensity = this._runtime.midIntensity * ((cfg.criticalPressure - pressure) || 0) / Math.max(0.01,
    (cfg.criticalPressure - cfg.midPressure));
270     this._runtime.targetIntensity = Math.min(Math.max(intensity, 0), cfg.maxMotorIntensity);
271 }
```



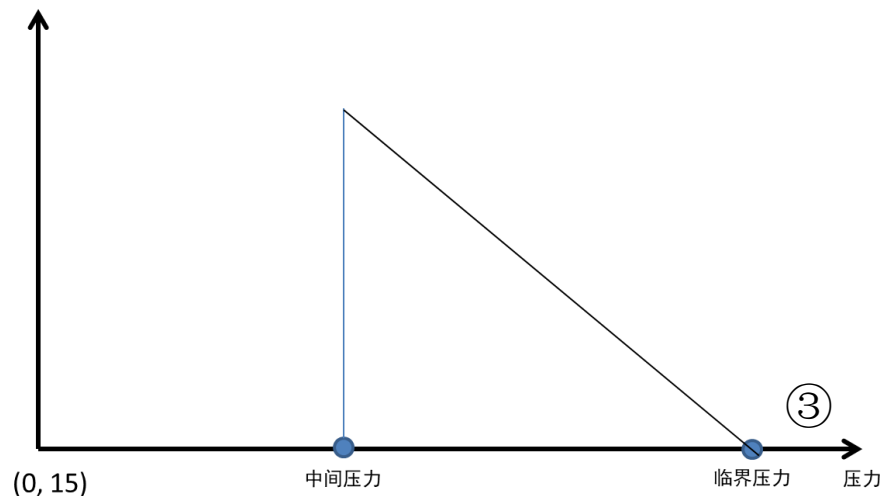
三阶段，压力大于或等于临界压力，延迟开始时间为现在，被电击（如果有），寸止次数加一，发送寸止信息文本，进入延迟期（冷静期），强度为零

```
209 // 超压，重置延迟倒计时
210 if (pressure >= cfg.criticalPressure) this._runtime.delayStartTime = now;
```

```
212 // 处于延迟期
213 if ((now - this._runtime.delayStartTime) / 1000 < cfg.lowPressureDelay) {
214
215     // 若之前不处于延迟期，电击，寸止次数加一，发送寸止信息文本
216     if (this._runtime.isInDelayPeriod == false) {
217         this._triggerShock(deviceManager);
218         this._runtime.totalDeniedTimes ++;
219         deviceManager.log('info', `超过临界压力，强度设置为0`);
220         deviceManager.emitUi({ fields: { statusText: '待压力降低...' } });
221         this._runtime.isPressureLow = false;
222     }
}
```

```
238 // 维持延迟期状态
239 this._runtime.isInDelayPeriod = true;
240 this._runtime.targetIntensity = 0;
241 }
```

文本一
(寸止)



延迟期的玩法逻辑和算法实现

进入延迟期：任何情况下导致的肛门压力超过临界压力。

本玩法的设计是，被寸止后，用户必须在低于临界压力的范围内等待一段时间（即用户设置的“lowPressureDelay”）让其冷静下来，才能开始下一游戏。

因此，

1. 只有当肛门压力低于临界压力时，才会开始进入低压延迟的倒计时”；
2. 如果在这倒计时期间肛门压力又达到或超过临界压力，那么之前的计时清零。

用户被跳蛋刺激导致肛门压力超过临界压力后，虽然跳蛋强度已经为零了，但肛门压力并不是瞬间降到临界压力之下的，需要一段时间。

所采用的实现办法：

只要肛门压力超过临界压力，延迟开始时间就一直被赋值为now，当低于临界压力时，延迟开始时间就不会被赋值为now。

对于开发人员：

1. 虽然延迟开始时间被赋值为now，但不意味着开始了延迟倒计时。
2. 延迟期不等于开始延迟倒计时

在这之前，delay start time被初始化为：

```
84 start(deviceManager, parameters) {
```

```
100     this._runtime.delayStartTime = (-1) * this._runtime.config.lowPressureDelay - 1e-6;
```

```
209 // 超压，重置延迟倒计时
210 if (pressure >= cfg.criticalPressure) this._runtime.delayStartTime = now;
211
212 // 处于延迟期
213 if ((now - this._runtime.delayStartTime) / 1000 < cfg.lowPressureDelay) {
214
215     // 若之前不处于延迟期，电击，寸止次数加一，发送寸止信息文本
216     if (this._runtime.isInDelayPeriod == false) {
217         this._triggerShock(deviceManager);
218         this._runtime.totalDeniedTimes ++;
219         deviceManager.log('info', `超过临界压力，强度设置为0`);
220         deviceManager.emitUi({ fields: { statusText: '待压力降低...' } });
221         this._runtime.isPressureLow = false;
222     }
223
224     // 之前低压现在超压，发送文本
225     else if (this._runtime.isPressureLow && pressure >= cfg.criticalPressure) {
226         deviceManager.log('info', `延迟期内超过临界压力，重置延迟倒计时`);
227         deviceManager.emitUi({ fields: { statusText: '待压力降低...' } });
228         this._runtime.isPressureLow = false;
229     }
230
231     // 之前超压现在低压，发送文本
232     else if (!this._runtime.isPressureLow && pressure < cfg.criticalPressure) {
233         deviceManager.log('info', `压力低于临界值，开始延迟 ${cfg.lowPressureDelay}s`);
234         deviceManager.emitUi({ fields: { statusText: `延迟期中(${cfg.lowPressureDelay}s)...` } });
235         this._runtime.isPressureLow = true;
236     }
237
238     // 维持延迟期状态
239     this._runtime.isInDelayPeriod = true;
240     this._runtime.targetIntensity = 0;
241 }
```

确保第一次运行时一定不会处于延迟期

延迟期内所推送的文本

isPressureLow: 上一次循环时肛门压力是否低于临界压力。
是为true, 否为false。初始化为true

```
98      this._runtime.isPressureLow = true;
```

延迟期内, 如果肛门压力达到或超过了临界压力, 而上一次循环时标记isPressureLow显示肛门压力低于临界压力, 说明这次循环是肛门压力首次超过临界压力的那一次循环, 发送延迟期内超压文本。

```
224      // 之前低压现在超压, 发送文本
225      else if(this._runtime.isPressureLow && pressure >= cfg.criticalPressure){
226          deviceManager.log('info', `延迟期内超过临界压力, 重置延迟倒计时`);
227          deviceManager.emitUi({ fields: { statusText: '待压力降低...' } });
228          this._runtime.isPressureLow = false;
229      }
```

文本二
(重置倒计时)

延迟期内, 如果肛门压力低于临界压力, 而上一次循环时标记isPressureLow显示肛门压力高于临界压力, 说明这次循环是肛门压力首次在临界压力之下的那一次循环, 发送开始延迟文本。

```
231      //之前超压现在低压, 发送文本
232      else if(!this._runtime.isPressureLow && pressure < cfg.criticalPressure){
233          deviceManager.log('info', `压力低于临界值, 开始延迟 ${cfg.lowPressureDelay}s`);
234          deviceManager.emitUi({ fields: { statusText: `延迟期中(${cfg.lowPressureDelay}s)...` } });
235          this._runtime.isPressureLow = true;
236      }
```

文本三
(开始倒计时)

延迟期结束，三个阶段开始

只有当now与delay start
time的差值大于或等于用
户设置的
lowPressureDelay时，
才会跳转到else中，即前
面的三个阶段。

与最开始游戏时强度从零
增加不同的是，
在游戏的第一阶段，即强
度逐渐提升阶段，强度不
会从零开始，即初始强度
不是零，而是根据当前的
肛门压力做出调整，若肛
门压力低，那么初始强度
会高一些，若肛门压力高，
那么初始强度会低一些。

```
243 // 不处于延迟期
244 else{
245     // 若之前处于延迟期，结束延迟期，计算初始强度，发送文本
246     if(this._runtime.isInDelayPeriod == true){
247         const baseTarget = cfg.maxMotorIntensity * (cfg.criticalPressure - pressure) / Math.max(1e-6,
248             (cfg.criticalPressure-cfg.pressureSensitivity));
249         this._runtime.targetIntensity = baseTarget;
250         this._runtime.unRandomIntensity = baseTarget;
251         this._runtime.isAtStartIntensity = true;
252         deviceManager.log('info', `延迟结束，基础强度: ${baseTarget.toFixed(1)}，开始逐步提升`);
253         deviceManager.emitUi({ fields: { statusText: '强度逐步提升中...' } });
254     }
255
256     // 低于中间压力，逐步提升目标强度，应用随机扰动，中间强度实时更新为当前强度
257     else if(pressure <= cfg.midPressure){
258         const increaseIntensity = dtSec * (cfg.intensityGradualIncrease || 0);
259         this._runtime.unRandomIntensity += increaseIntensity;
260         const rnd = 1 + (Math.random() - 0.5) * 2 * (cfg.stimulationRampRandomPercent / 100);
261         this._runtime.targetIntensity = Math.min(Math.max(this._runtime.unRandomIntensity * rnd,
262             0), cfg.maxMotorIntensity);
263         this._runtime.midIntensity = this._runtime.currentIntensity;
264     }
265
266     // 介于中间压力和临界压力之间，
267     // 以上方if最后抓取的midIntensity作为midPressure的压力，
268     // 将目标强度以当前压力进行以点(midPressure, midIntensity)和点(criticalPressure, 0)之间连线的线性映射
269     else if(pressure < cfg.criticalPressure){
270         const intensity = this._runtime.midIntensity * ((cfg.criticalPressure - pressure) || 0) / Math.max(0.01,
271             (cfg.criticalPressure-cfg.midPressure));
272         this._runtime.targetIntensity = Math.min(Math.max(intensity, 0), cfg.maxMotorIntensity);
273     }
274
275     //维持非延迟期状态
276     this._runtime.isInDelayPeriod = false;
277 }
```

文本四 (基础强度计算)

初始强度算法

刚开始游戏时，初始强度被初始化为零

```
93     this._runtime.unRandomIntensity = 0;  
94     this._runtime.targetIntensity = 0;
```

以后的游戏中，延迟期结束，开始计算初始强度的算法为

```
248     const baseTarget = cfg.maxMotorIntensity * (cfg.criticalPressure - pressure) / Math.max(1e-6,  
    (cfg.criticalPressure - cfg.pressureSensitivity));
```

对于开发人员：

图和代码中的压力敏感系数

pressureSensitivity实际上是“用于计算初始强度的参考点”，单位应为kPa。显然不是什么系数，因为“系数”是没有单位的。

首先，这个“用于计算初始强度的参考点”不能超过临界压力。

其次，从图中可以看出，这个值越大，映射斜线越陡峭，意味着当前压力对于初始强度大小的影响很大。

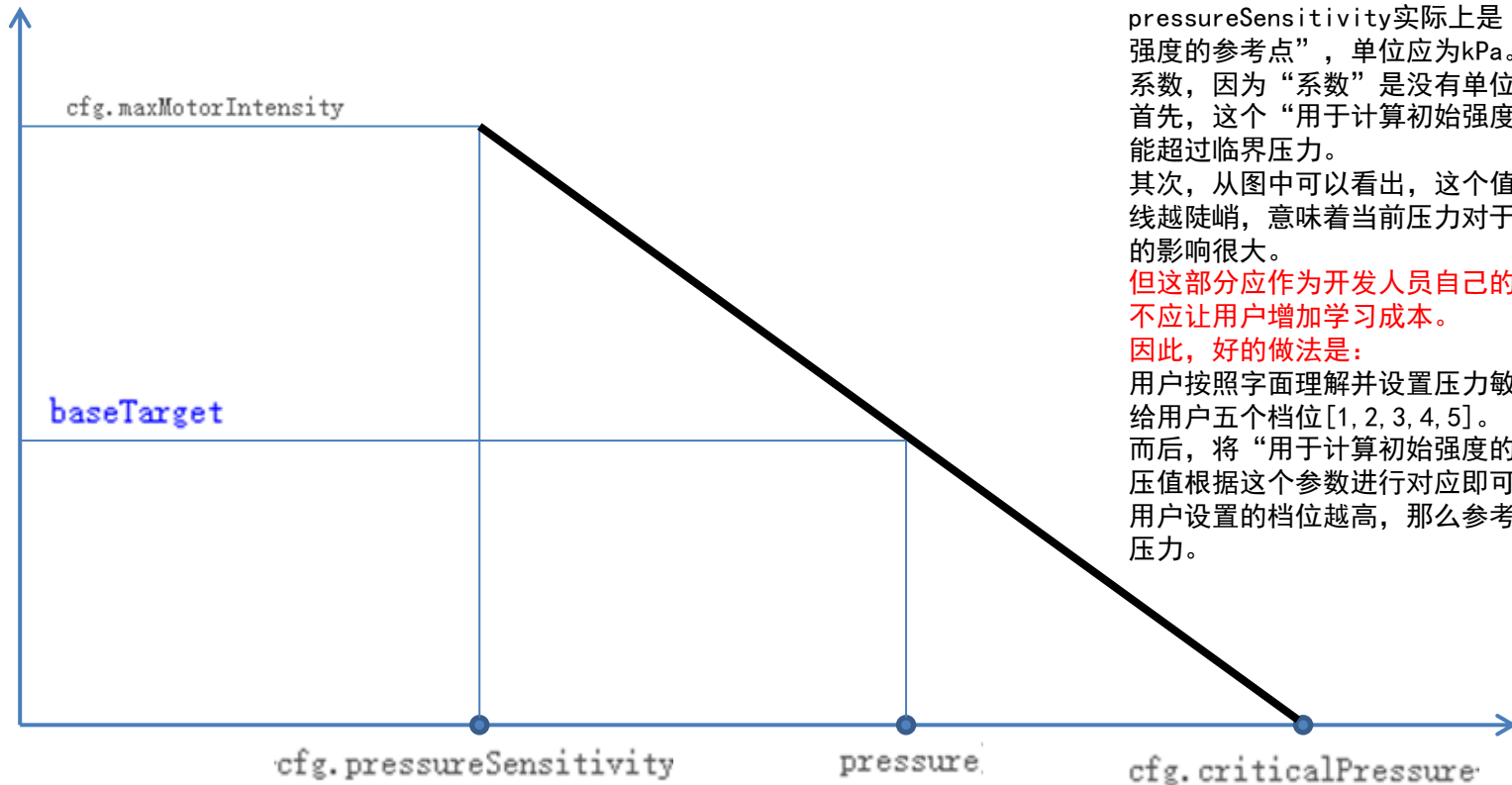
但这部分应作为开发人员自己的知识就好，不应让用户增加学习成本。

因此，好的做法是：

用户按照字面理解并设置压力敏感系数，提供给用户五个档位[1, 2, 3, 4, 5]。

而后，将“用于计算初始强度的参考点”的气压值根据这个参数进行对应即可。

用户设置的档位越高，那么参考点越接近临界压力。

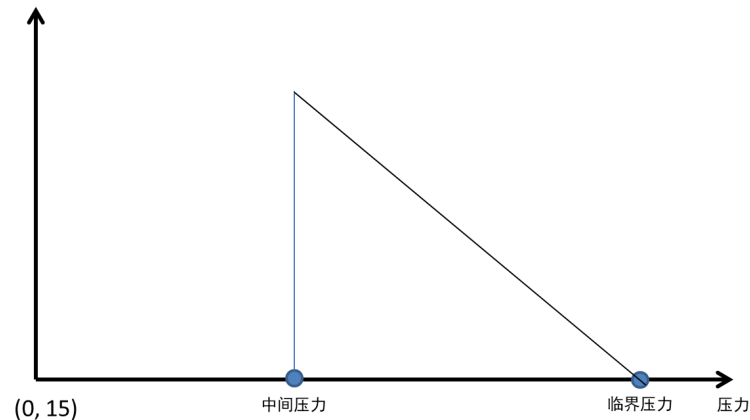
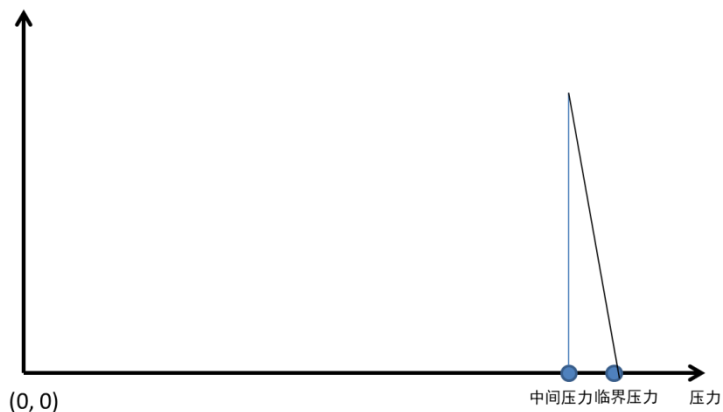


UI画面要注意的事项

人体肛门压力一般维持在**17-21kPa**之间，游戏开始后压力不可能降到很低的水平。因此没有必要在画面中显示出明显低于人体肛门的压力。

若从压力为零为原点（左图），会很不美观中间压力和临界压力的距离很少，并且都挤在右边，用户用不方便用拖拽的方式调节。

可以从**15kPa**开始或使得图更美观的压力作为动态图的原点。



关于游戏过程中推送的文本

文本一
(寸止)

文本二
(重置倒计时)


文本三
(开始倒计时)

文本四
(基础强度计算)

目前来说太机器化了，可以多一些设计，增加用户趣味。

关于体验的其他事项

```
131         this._runtime.averagePressure = recent.length ? (sum / recent.length) : p;  
132         // 将最新压力发给 UI  
133         deviceManager.emitState({ currentPressure: p, averagePressure: this._runtime.averagePressure });
```

- 
- 用户是否有必要自己手动设置临界压力？
 - 既然我们能分析出平均压力，不如将这个压力往上增加一些值作为临界压力，是不是更“智能”？
 - 输入对于用户体验而言是一个辩证的关系。有时用户设置的越少，就越方便，就越智能；有时用户想在游戏过程中实时去调整一些值，而不必停止整个正在进行的游戏进程，才显得更方便更智能。
 - 因此，哪些是开始游戏前避不开的，必须由用户设置的，我们才不得不让他来设置；哪些是我们可以根据已知信息自动调整的，我们就坚决不要麻烦用户来设置；哪些是用户希望游戏过程中能实时调整的，我们就给他做一个在游戏过程中能够实时调整的按钮。
 - 这个按钮，可以是按钮，也可以是一个可拖动的点。这就涉及到人机交互体验的问题了……