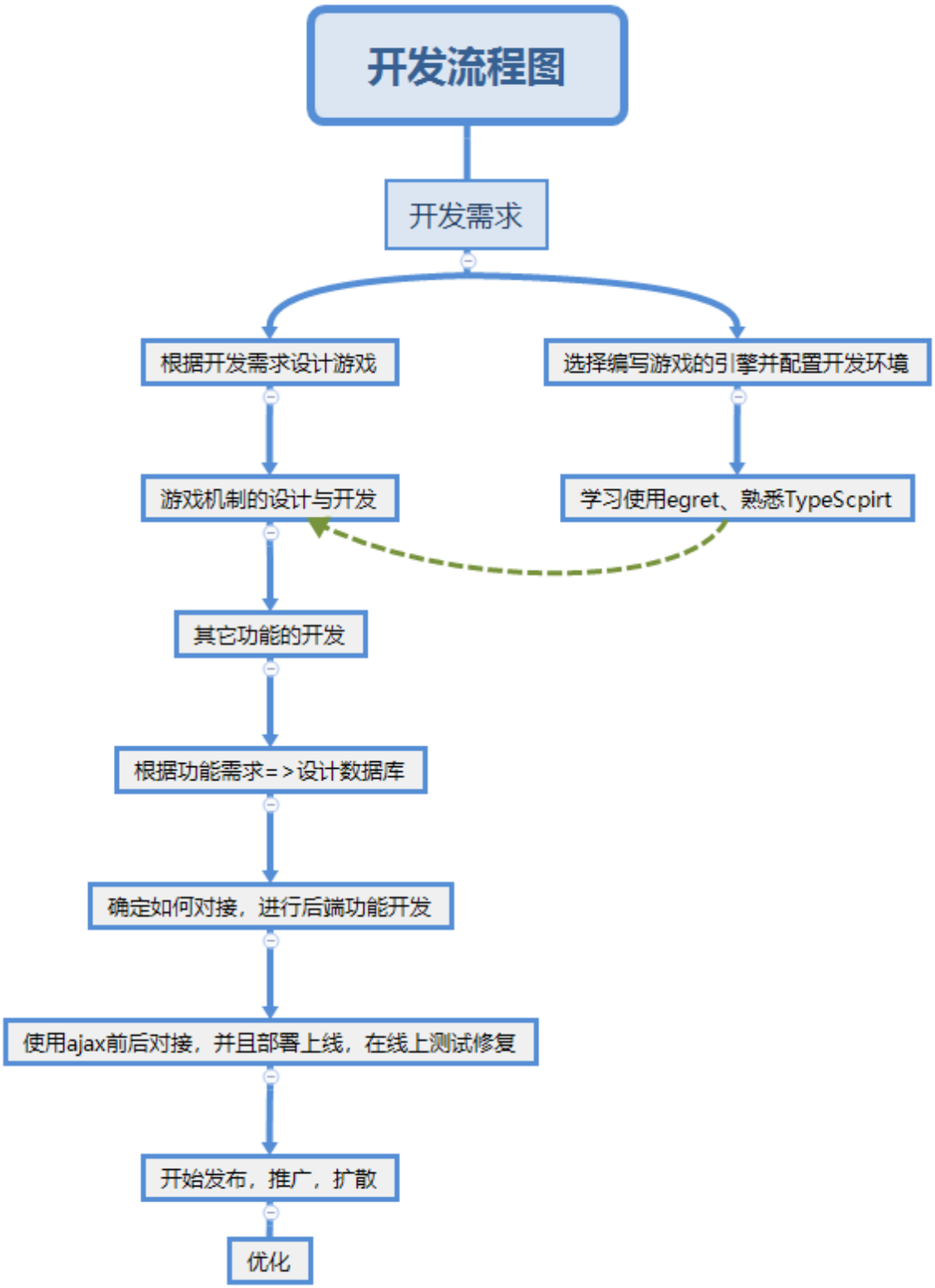
开发思路



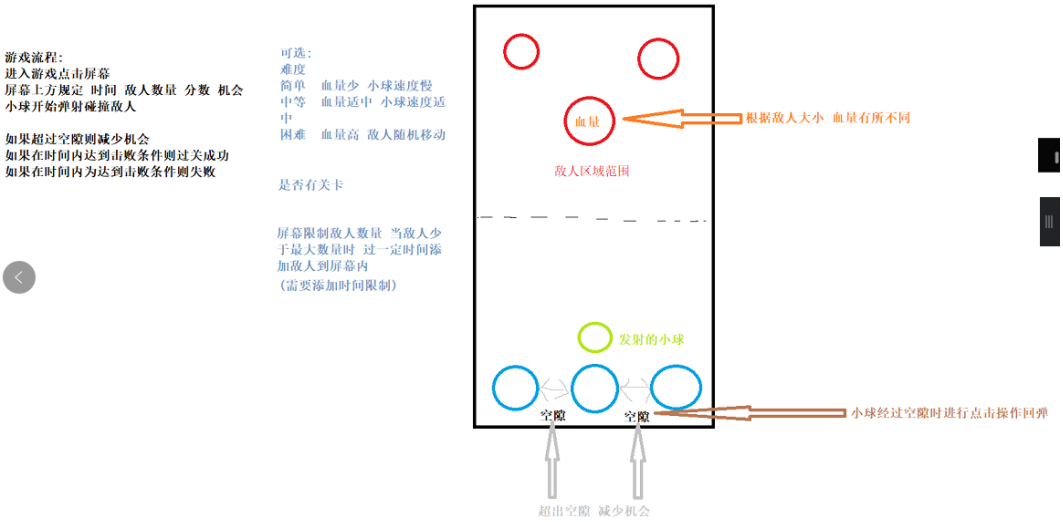
1. 开发需求：

   根据新网银行提供的动画素材，研发一款简单好玩的弹珠类游戏

2. 根据需求设计游戏：

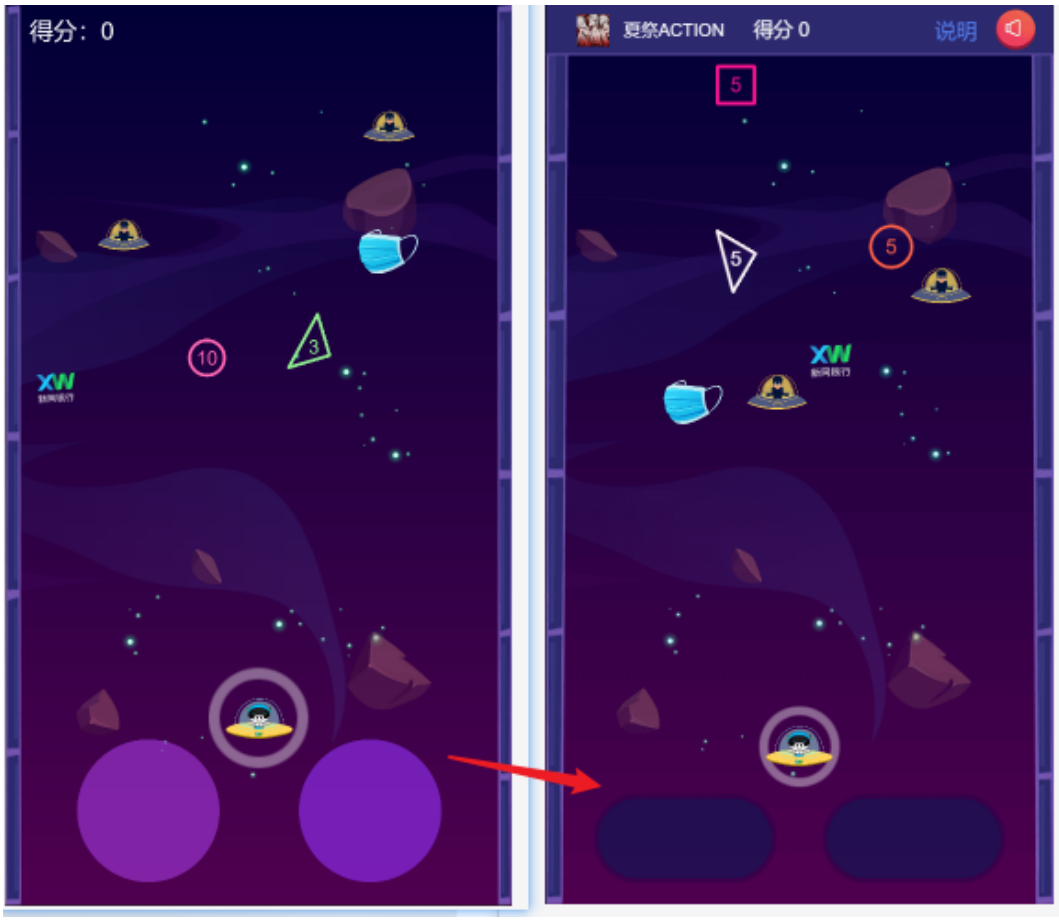
- 由于官方提供的动画素材比较偏向于 “飞船、星空” 的设计元素，所以经过我们队伍一起讨论过后，希望在“朱望仔大战大反派”为主题的基础上，延伸成 “星空大作战” 这样更广阔，新颖的主题，一个带有外太空背景的游戏主题。于是，进入游戏后的映入眼帘的标题也起名为“星空大作战”。
- 官方赛题要求：弹珠类游戏

  我们团队在各大游戏平台（例如4399小游戏等）体验了各种各样的弹珠类游戏，并且想到了当年很经典的windows附带的一款电脑游戏《三维弹球》，结合我们的体验及对《三维弹球》这款经典的游戏情怀，设计了《星空大作战》这款弹珠类游戏（玩法：游戏页面下方设置玩家操控的“动态挡板”，游戏页面上半部分主体中放置一些多边形的碰撞物体，以及类似官方Demo游戏中的得分物体，玩家如果没有成功防止朱望仔落出屏幕下方，即视为游戏挑战失败）



  当然，这个图只是我们最初的设计雏形，后期为了优化游戏体验，做了一些调整，改变。

  例如从下方的回弹物体从圆球，调整为按钮和光线区域



3. 选择编写游戏的引擎 并 配置开发环境

在有了大概的游戏设计思路后，我们开始寻找适合我们的游戏开发引擎，由于我们对`JavaScript`语言最为熟悉，所以在多个小型游戏开发引擎中选择了：`egret`白鹭。因为在egret引擎中，主要使用的是`TypeScript`语言，我们已经熟知`TypeScript`是`JavaScript`的一个超集，这样，我们只要在`JavaScript`的基础上进一步学会`TypeScript`的语法以及了解两者主要的差异后，就可以开始用`TypeScript`直接上手在`egert`游戏引擎中编写代码；并且官方要求是微信上可以玩的小游戏，egret游戏开发引擎，算是小游戏开发领域中还算主流的引擎，有不少典型小游戏的案例，有助于让我们的开发过程稳定高效。

4. 学习使用开发工具

   由于我们之前从未有过开发游戏的经验，也就从来没用过这些游戏引擎，而且虽然比赛是在6月17日开始，但是我们实际上是7月22日才得知比赛并开始准备，在选择好游戏引擎 并 决定用主办方提供的动画素材，从 0 开始开发一款弹珠类游戏后，我们第一时间做的事就是学习`TypeScript`和`egret`引擎中的基本概念，不断地查文档、看视频去学习我们可能需要用到的知识点，并在学习过程中通过练习去熟悉`egret`中编写游戏的思路、经常用到的`API`，在经过大概4天的时间，我们感觉通过学习到的各方面的知识差不多已经可以实际开始编写游戏，然后我们开始制作开始游戏的第一个游戏页面，后期遇到没学到过的盲点，大多都是遇到后通过查文档来遇到一个解决一个。

5. 游戏体验机制的设计与开发

   作为一款游戏，我们不想随随便便做一款自己想做的游戏（不想完全按自己的意愿来），我们想更多的考虑什么样的游戏能给用户带来吸引力，有些刺激但又在掌控感之内，换句话说：我们想做能让玩家体验到乐趣、刺激的游戏。

   我们觉得最重要的是游戏过程要吸引用户，让用户的注意力随着游戏难度提升而需要越来越集中，并且让游戏中具有随机性的元素。由此，我们设计了以下几点游戏核心：

   - 引用了`p2`物理引擎，首先设定好一个不会有速度衰减且有边界的物理世界，然后通过玩家点击下面按钮，对 “朱望仔” 施力，使其在屏幕上半部分中与各种物体物理碰撞。

   - 让玩家越来越集中

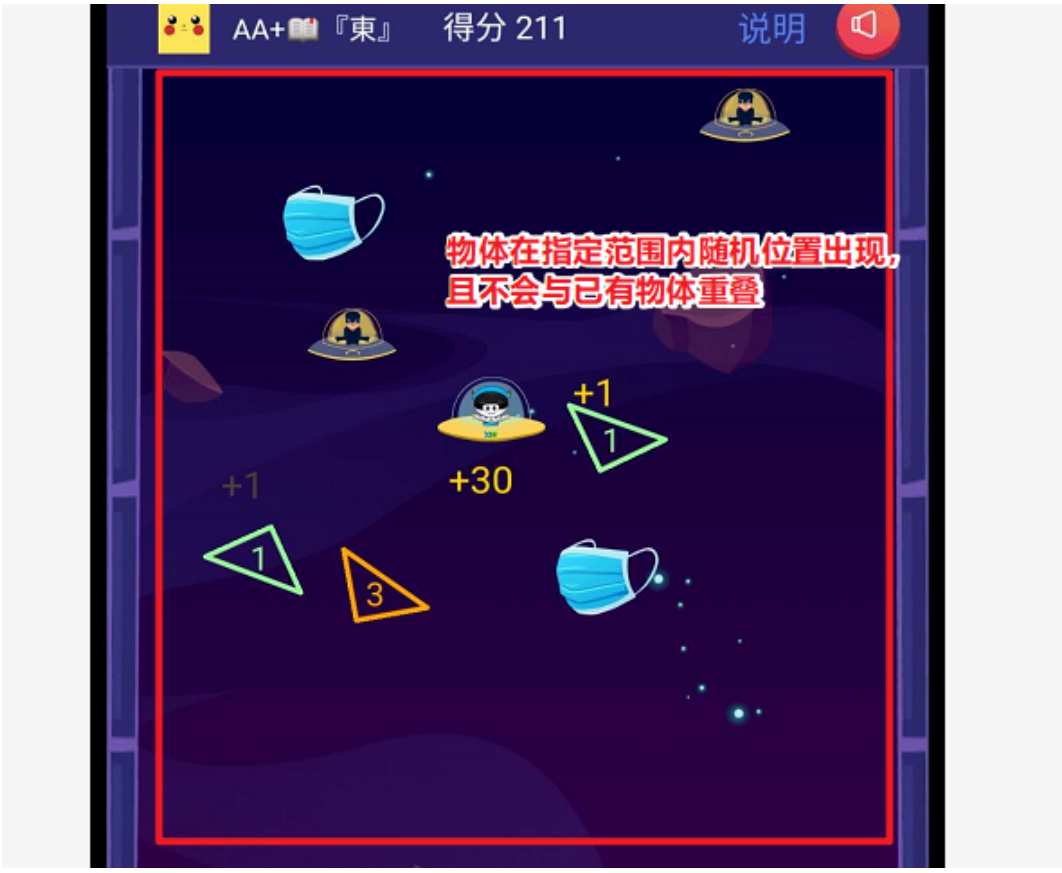
     “朱望仔” 在游戏进程越到后期时，在掉下后会被重新施加冲力（一次比一次的力大），移动速度越来越快，这样，玩家才能感受到游戏难度的逐步提升，并且下意识地越来越集中注意力。具体的实现方式，后面会在优化中更详细的说明，或者代码中有注释。

为了更好地控制速度变化，通过数学中的向量知识点，使其不管朝哪个角度回弹，x轴和y轴合起来的最终速度向量大小都和在我们要实现的大小一致。

- 随机性

  在“朱望仔”消灭掉多边形和吃到得分物体（这里指口罩、大反派、新网银行logo）后，游戏上半部分区域中会在随机位置，生成新的随机物体。例如：如果随机生成一个正方形并且随机生成的位置比较接近下面的“挡板”，那么“朱望仔”很可能在回弹时撞到正方形，直接回弹到下面，让玩家产生猝不及防的紧张感。

  而且我们使用了数学距离公式，`es6` 的every方法结合while循环，来实现每次生成的随机位置，不会和现有的物体位置重叠遮盖。



- 平均每局游戏从开始的简易，到后期的高难度的时长把控

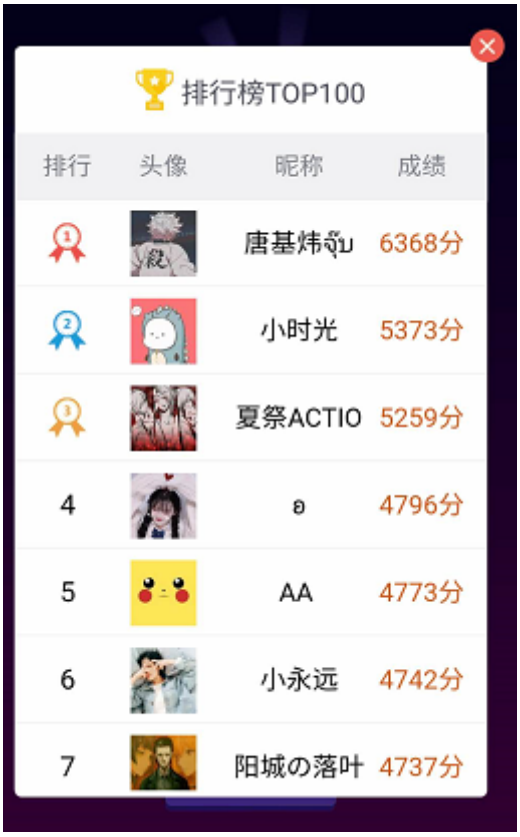
为了更好的把控游戏难度极限到来的时机，我们团队成员一起玩了一些经典的微信小游戏，例如《欢乐球球》，我们正常的去玩这些游戏，并且记录我们每句的平均游戏时长，得到我们普通玩家平均每局游戏的时常大概在1分钟30秒的时长左右，所以我们《星空大作战》的游戏最终难度就通过各种维度把控在了大概将近2分钟的左右，而且到了游戏后期 “朱望仔” 的速度已经很快的时候，减小了难度提升的幅度，让玩家后期还有一定掌控感。

- 为了激励玩家每局游戏的坚持

  我们通过间接检测 “朱望仔” 的速度，使上方区域随机生成的得分物体越来越多，而且与多边形每次碰撞的得分也会越来越多，这样也就意味着，虽然后期速度很快，但是得分效率也很高。

6. 其他功能的开发

- 排行榜功能，参照官方demo游戏的排行榜功能，帮助游戏能更好的扩散，也刺激了玩家的追求更高排名的心理



- 带有二维码识别的生成成绩单功能：

  为了更好的回馈我们的主办方，我们也制作了生成成绩单的功能，并且在生成成绩单页面中的下面，放上了主办方的公众号二维码，由于游戏引擎中的图片不能直接被微信长按扫码的功能支持，我们通过在 `html` 中加入 `img` 标签的方式实现了在微信浏览器上可以让 新网银行公众号的二维码 可以被用户长按识别的功能。

- 游戏说明提示的开发

  我们通过 `localstorage` 记录用户是否已经阅读过游戏说明内容，如果用户是第一次进入游戏，会自动弹出游戏说明；如果用户已经知晓了游戏说明，以后再次进入游戏后便不会再进行不必要的游戏说明自动提示。

- 游戏音效

  为了让玩家有更好的游戏体验，我们在开发过程中添加游戏的按键音效和得分音效。并且也用`localstorage`记录了玩家是否有打开音效的习惯（如果玩家上次是打开音效玩游戏，下次进入游戏时，音效会自动打开，相反，如果玩家上次没有打开音效，以后打开游戏，音效也不会自动开启。）

7. 数据库准备

   在基本的游戏部分，其它功能部分都做好后，进行数据库及其数据业务流程的分析，设计了`mysql`数据表

8. 后端功能开发

   根据数据表的设计，以及游戏部分需要的排行榜需求，保存用户信息的需求，更新用户分数的需求，使用`nodejs`的`express`框架进行了后端搭建

   - 获取用户信息功能:
     在玩家进入游戏中 ,用户的名称,头像,以及登入时间等记录就会在数据库中储存，并且判断用户的id是否已经存在，如果已经存在进行更新用户数据（因为有些用户的头像更新后，我们也要随之更新新头像），如果没有就添加数据
   - 获取用户的得分功能:
     当玩家在游戏中通过猪望仔碰撞到得分物时,在游戏的上方会显示用户一共所得的分数,并且这个分数在游戏结束后 后台将会判断是否为玩家所得的最高分数,如果是,那么恭喜这位玩家又

突破了自己所有的记录,并且数据库中会自动储存这位玩家的最高分,这样就可以方便玩家查看下一次是否又突破了自己
    - 排行榜功能:
      当有大量用户游玩后,为了能方便看到不同玩家所获得的成绩,我们设计了排行榜功能,并将玩家获得的分数从高到低依次排序,我们希望玩家可以通过这种方式能激励玩家进行下一次的游戏,如果在排行榜中并没有看到您自己,那么请再接再厉,因为我们只通过数据库拿出100条数据
    - 通过用户分数，获取用户超越了百分之N的玩家功能:
      玩家在游戏结束后,会看到这位玩家获得的分数超过百分之多少的其他玩家,我们通过数据库获取到这位玩家分数超过的人数和总玩家人数 计算得出百分比,并在游戏结束后显示在窗口中来激励玩家下一次的游玩
9. 线上测试修复

   在得到官方域名的映射链接后，使用`nginx` 让用户可以通过80端口访问到游戏的静态`index.html`文件，然后使用 `nginx` 反向代理将 `/get` 指向 服务器的3000端口，同时在服务器上启动了3000端口运行游戏后端，在线上测试发现了后端写好的代码中，存在时间格式不对应，`sql`语句不完善的问题，及时修复后，进行发布推广，扩散
10. 发布后的2~3天时间，在身边体验过游戏玩家的各方面建议下，不断对游戏各方面进行优化，补充，调整。

营销手段

1. 主要是通过朋友圈在老师，同学的帮助下进行推广，扩散
2. 在一些小游戏的广播聊天中，发散过
3. 鼓励玩家互相进行打榜比拼

优化工作

1. 优化加载
    - 优化原因：我们对于进入游戏时的加载不是很满意，想让加载时速度提快点
    - 优化过程
        - 我们通过网上查资料找到了一些让项目“瘦身”的方式
        - 借助`egret`白鹭的工具Texture Merger 可以让我们所有分开的图合成一张大的精灵图，由于图片的大小取决于图片中颜色的复杂度，所以这样处理后可以减小不少,但我们还是不满意
        - 于是继续对该大图进行一个合理压缩，同时也要保证图片在游戏内的美观，无法保证美观的就单独一个图放着
    - 优化结果：加载时间至少是之前加载时长的1/2，甚至更短，即便是第一次加载
2. 优化排行榜
    - 优化原因：排行榜在初期做好了 `TOP100` 数据的时候，点击后需要等100条数据都渲染进去才会整个弹出展示给玩家看，这样就造成点击排行榜后会在黑色透明遮罩下有一段等待期
    - 优化过程
        - 在点击排行榜后先弹出排行榜的框架，然后在拿到数据后将数据一行一行加入滚动视窗
    - 优化结果：没有了之前那么明显的等待期，先出现白色窗口，数据也比较快速渲染进去
3. 优化游戏分享
    - 优化原因：游戏域名需要借助官方提供的链接中转才能获取到微信用户信息，可是获取到信息的同时，链接的末尾也会带上该用户的信息，我们发现如果该用户进入游戏链接后，选择微信右上角的三个点分享出去的话，就会自然而然带上自身的信息，那么其他用户如果通过该分享出来的链接进入的话就会发现游戏内不是自己的头像与昵称，而是分享者的!
    - 优化过程

        - 在老师和学长的帮助下，学习到有个很巧妙的办法可以解决这个问题
        - 在前端页面判断是否拿到了用户信息，没拿到就通过官方链接中转拿信息，直到拿到之后将信息存到内存数据，并直接跳转到我们的游戏域名链接
        - 拿到我们存好的信息数据进行渲染信息等操作
        - 那么如果用户再进行分享的话，就是直接通过我们的游戏域名进行分享，不会带上个人的信息，这样就可以实现安全分享
    - 优化结果：实现安全分享

4. 游戏难度进展的控制
    - 优化原因：考虑到不同程度的玩家 玩家实力也不尽相同，也为了让玩家有更好的游戏体验
    - 优化过程
        - 主要的难度设置就是“朱望仔”会随着点击按钮后的碰撞逐渐加速
            - 设定了三个阶段的加速：前中后期
            - 前期到中期较于中期到后期的进度较快，让玩家处于中期的时间较长
        - 我们需要设定初始速度值，初始速度变化值，通过调试判断出前期到达中期和中期到达后期分别所需要的的碰撞次数，通过判断点击按钮后的总碰撞次数对每次的速度递进值进行修改（前期速度变化值 > 中期 > 后期）
        - 在调试记录碰撞次数的时候，发现快速点击按钮会一下触发多次碰撞次数的累加bug。解决方案：
            - 设定一个节流阀(初始为true)，为true才能发生让”朱望仔“碰撞回弹
            - 点击按钮和“朱望仔”发生碰撞回弹后，关闭节流阀(false)，判断朱望仔已经回弹离开光线，再开启节流阀(true)，这样就解决了 让过度点击按钮不会让“朱望仔”发生发生多次重复碰撞，同时也让我们可以更精准把握游戏难度
5. 优化后期游戏难度太高造成的失控感
    - 优化原因：在后期 “朱望仔” 速度基数很高时，如果遇到比较低的多边形，很容易在眼神都跟不上的瞬间回弹到下面，导致游戏失败
    - 优化过程：
        - 将随机生成一个新位置的函数中的区域范围的y轴改为一个可以通过this访问到的变量，在后期速度很快时，讲y轴的区域范围提高，这样就不会在太低的地方生成多边形导致游戏难度过高，让玩家有耐心继续坚持下去
6. 回弹角度
    - 优化原因：原本朱望仔碰到下面光线回弹时，角度范围在 45度-90度 之间，x轴正反方向随机，但是感觉角度随机的范围有些小
    - 优化过程：通过勾股定理的公式，让角度随机范围 在30度-90度 之间，让角度变化的随机性更明显
7. 生成多边形时，不能碰撞到“朱望仔”
    - 优化原因：在`p2`物理引擎中，如果世界中的刚体进入世界时，刚好遇到了 “朱望仔” 活动的位置，会让 “朱望仔” 明显减速或加速
    - 优化过程：在随机生成位置的时候，获取朱望仔的位置，也加入到数组中使用every方法比较距离判断，判断位置是否可取，可取才生成
    - 优化结果：在游戏前中期已经完全避免，不过游戏后期，速度很快时，还是小概率会发生刚体相叠导致加速或减速。
8. 游戏结束后的优化
    - 优化原因：在游戏结束后，会发起post请求查询用户得分超越多少人，且总共有多少人有分数，但是这里是一个异步的过程，如果有些玩家很快的去点击生成成绩单，成绩单中的超过多少人就会显示undefined
    - 优化过程：所以我们在游戏结束后，先让生成成绩单的按钮不可点击，等异步的`ajax`请求返回结果后，我们再给按钮开启可点击的功能，生成成绩单。

- 优化结果：生成的成绩单中不再可能会出现undefined。

9. 减少游戏过程中不断占用的内存
    - 优化原因：我们在游戏过程中会声明很多变量来暂时保存对象，但是这些变量越来越多的时候，会大量占用游戏内存，导致游戏变卡
    - 优化过程：检查所有代码过程，在使用完临时对象，添加到舞台或世界后，将声明的变量赋值为null