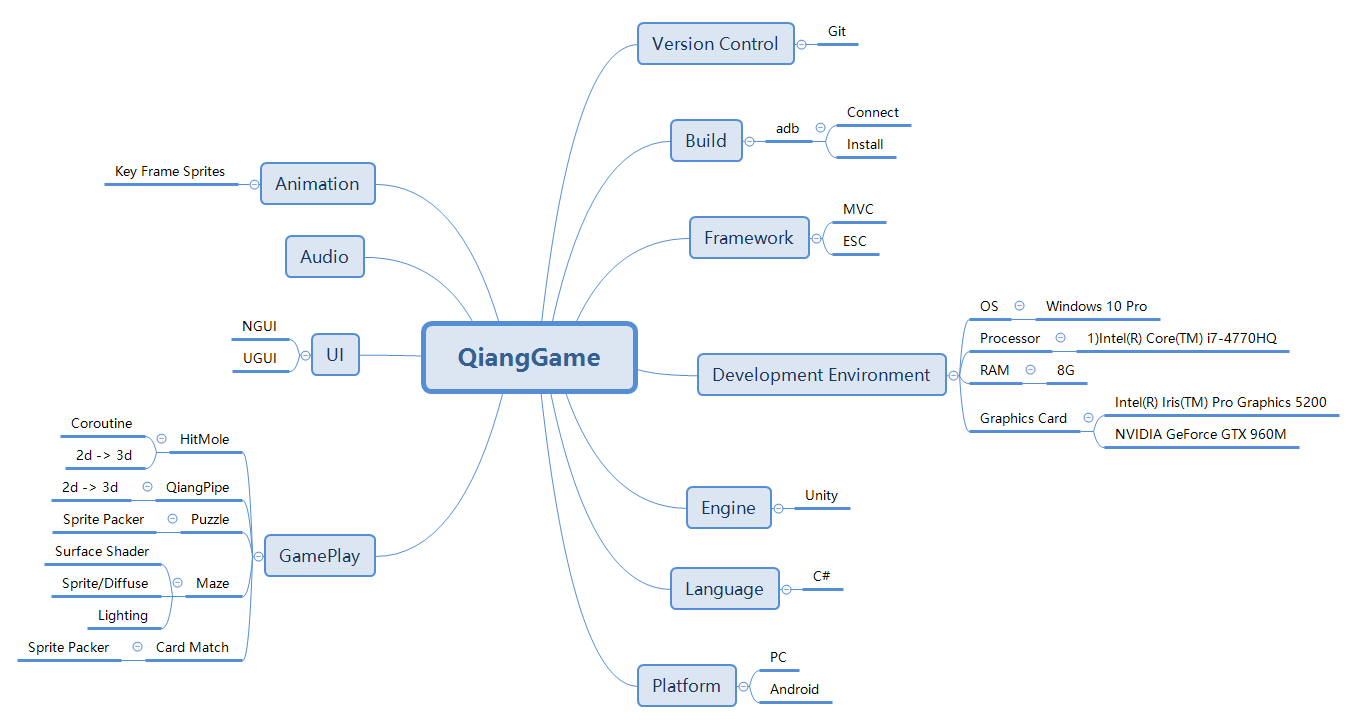
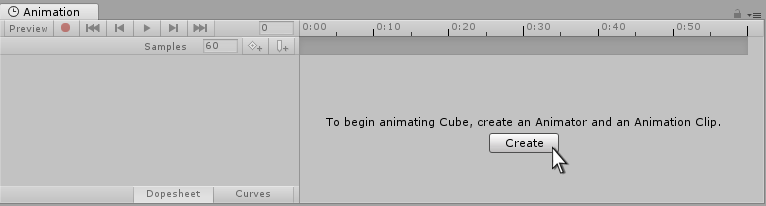
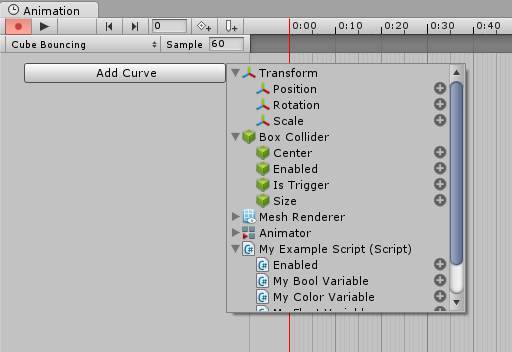
1. 系统整体框图：



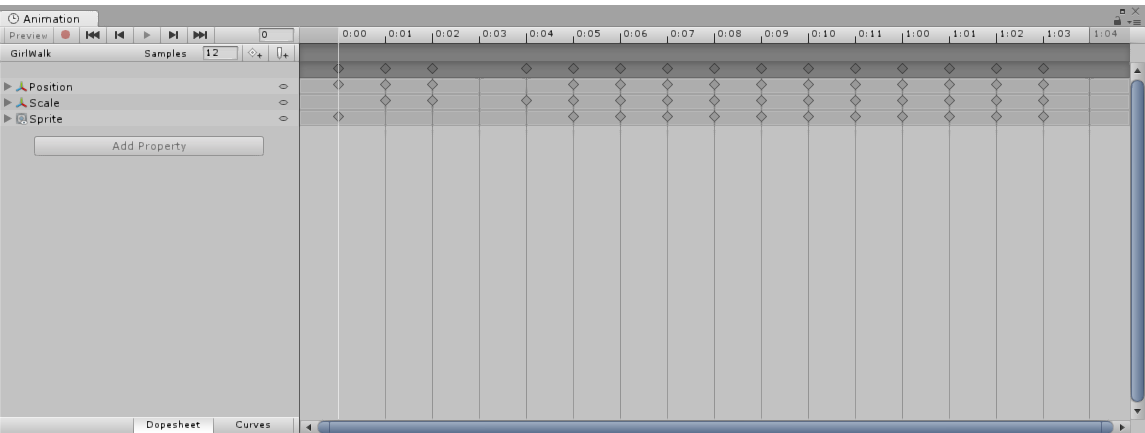
1. 模块的设计与实现：
   1. 动画：
      1. 设计：
         1. 实现方式：通过Unity自带动画编辑器，利用图片序列制作帧动画，并用脚本对动画状态进行控制；
         2. 所需资源：帧动画图片序列
      2. 实现：
         1. 创建动画剪辑(Animation Clip)：
            1. 在 Unity 中，为了让游戏对象动起来，需要附加一个动画组件。这个动画组件必须引用一个动画控制器，动画控制器再引用一个或多个动画剪辑。在 Unity 中，当开始使用动画视图让游戏对象动起来时，这些元素将被自动创建和绑定。
            2. 点击 动画视图(Animation) 左上角的下拉框，选择 Create New Clip，可以为选中的 游戏对象 创建一个新的 动画剪辑。然后，会提示你把动画剪辑保存到 Assets 文件夹的某个位置。如果这个游戏对象已经附加了动画组件，并且指定了动画控制器，新的动画剪辑将作为一个状态，被添加到现有的动画控制器中。



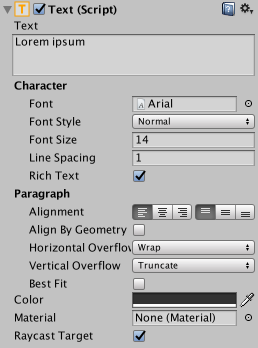
* + - 1. 使游戏对象动起来：
         1. 添加关键帧：点击动画记录按钮，进入动画记录模式，此时，对该游戏对象的修改会被记录到动画剪辑。通过再次点击动画模式按钮，从而退出动画记录模式。这一操作将把该游戏对象的状态恢复到进入动画记录模式之前。



* + - * 1. 组成图片序列：将添加的关键帧组成图片序列，点击播放按钮，则可预览当前动画剪辑的动画效果。

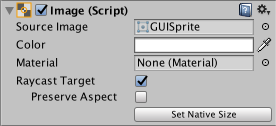


* 1. UI：
     1. 设计：
        1. 实现方案：以UGUI为主，对界面进行布局、编辑与屏幕自适应；NGUI为辅，实现DoTween、iTween等常用缓动效果与简单动画。
        2. 所需资源：UI美术素材及原型图。
     2. 实现：
        1. 视觉组件：
           1. 文本组件(Text):



文本 Text 组件（也称为标签 Label）具有一个文本区域，用于输入将要显示的文本。可以设置字体、字体样式、字体大小，以及是否具有富文本功能。

* + - * 1. 图像组件(Image)：



图像 Image 具有一个矩形变换组件 Rect Transform 和一个 图像 Image 组件。图片精灵可以被应用于图像组件的 Target Graphic 域，颜色可以在 Color 域设置。材质也可以应用于图像组件。Image Type 域定义了图像精灵的显示方式。

* + - * 1. 原始图像组件(Raw Image)：图像组件 Image 接受一张图像精灵，而 原始图像 Raw Image 接受一张纹理（无边框等）。原始图像只应该在必要时使用，大多数情况下，图像更适合。
        2. 遮罩(Mask)：遮罩 Mask 不是可见的 UI 控件，而是一种修改控件子元素外观的方法。遮罩限制（则遮盖）子元素为父元素形状。因此，如果子元素大于父元素，那么只有位于父元素中的部分将是可见的。
      1. 交互组件：
         1. 按钮 Button



按钮具有 OnClick 事件，用于定义点击后执行的行为。

* + - * 1. 开关 Toggle



开关 Toggle 含有一个复选框，用于确定 Toggle 当前是打开还是关闭的。当用户点击 Toggle，它的值被反转，并且相应的打开或关闭视觉标记。它还具有一个 OnValueCHanged 事件，用于定义值被改变时执行的行为。

* 1. GamePlay
     1. “打地鼠”游戏：
        1. 设计：
           1. 2d向3d视角的转换：通过对“地鼠”与“鼠洞”的图片切割，并更改其sortingLayer属性，表现出“地鼠”从“鼠洞”穿出的透视效果。
           2. 生成规则："地鼠"每隔一段时间，随机从所有鼠洞中的某一个鼠洞位置出现。
           3. 触发条件：

击打检测：通过用户输入（鼠标点击/手指触碰）确定击打位置，并通过击打位置判断此次击打是否有效；

击打规则：若击打有效，则根据锤子击打到不同类型的"地鼠"，分别给出不同响应：打到好人，扣除相应分数；打到坏人增加相应分数；若击打无效，不作反馈处理。

* + - * 1. 胜负判断：在一定时间范围内，分数是否到达胜利条件（一定分数值），若是，则游戏胜利，出现下一关入口；否则游戏失败，出现重玩入口。
      1. 实现：
         1. 2d向3d视角的转换：



图(1): 地鼠洞.sortingLayer = 0，角色.sortingLayer = 1

(角色层次高，显示在地鼠洞之前)



图(2): 在图(1)基础上，新增一个sortingLayer == 2的”半个地鼠洞”图片



图(3): 将其拼接至适当位置，即可实现”地鼠穿出”的效果

* + - * 1. 生成规则：

对每只“地鼠”，先根据一定概率随机出其“地鼠”类型；

再随机出其出现位置——“鼠洞”序号；

向上运动——钻出“鼠洞”；

随后停留一段时间供玩家进行操作响应；

向下运动——钻回“鼠洞”；

回到步骤a)，开始下一轮行为

* + - * 1. 触发条件：

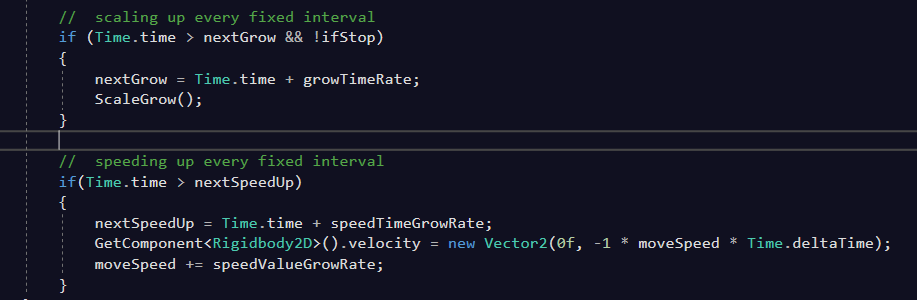
射线检测：获取用户点击屏幕的屏幕坐标，转换成世界坐标，再以该世界坐标点为起点，向z轴（正前）方向发出一条射线，并返回与该射线相交的游戏物体；

击打判断：若返回的游戏物体是“地鼠”物体，则通过获取该被击打“地鼠”的id判断击打到哪一只“地鼠”。

* + - * 1. 胜负判断：

定义有效时间、胜利标准分值，在有效时间内分数达到标准分值则获得游戏胜利，否则游戏失败。

* + 1. 羌笛音乐游戏：
       1. 设计：
          1. 2d向3d视角的转换：由于游戏中音符由远至近运动，根据“近大远小”的透视原理，通过改变音符在距离不同状态下的尺寸与运动速度，实现透视效果。
          2. 生成规则：以各自对应的概率分别生成三种类型的音符：长单音符、短单音符、双音符；生成的音符自动持续向下运动。
          3. 触发条件：在音符位置进入击打判定区域后，检测用户输入，若在有效区内完成输入，则根据音符类型增加对应的分值。
          4. 胜负判断：在指定时间内得到一定分值，判定游戏胜利；在游戏胜利条件下，再根据分值进行等级划分，对玩家进行评价反馈。否则游戏失败。
       2. 实现：
          1. 2d向3d视角的转换：



如上图代码示例，每隔一定时间间隔(growTimeRate, speedTimeGrowRate)，分别对尺寸和速度(沿-y轴方向)进行一定比例的增大，由于音符物体始终在朝-y方向运动，随着时间的推移，位置由远至近，与此同时，尺寸由小至大、速度由慢至快，实现近大远小、近快远慢的效果。

* + - * 1. 生成规则：

先计算概率，根据概率值，确定生成音符的位置 a. 左边 b. 中间 c. 右边

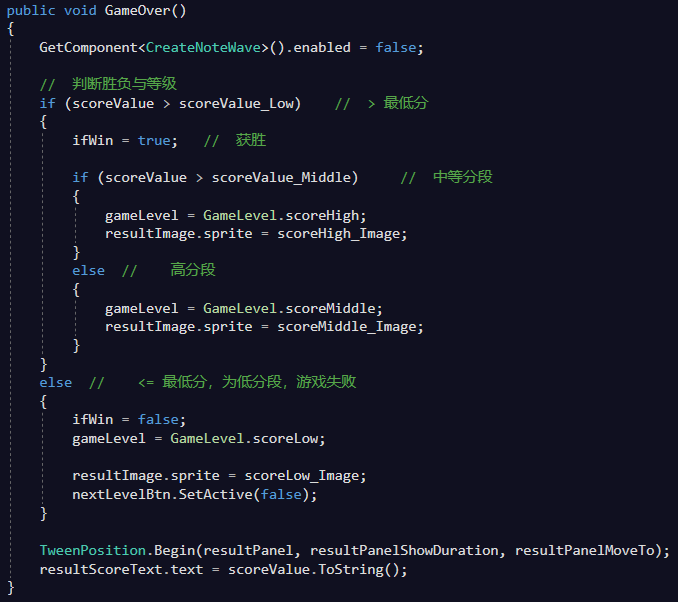
如果是 1.a 或 1.c，再确定生成音符的类型 a. 短音 b. 长音，如果是 1.b，则只有一种情况，即 c. 双音

确定了从某位置生成某类型的音符后，通过该类型音符预制体的引用将其实例化。

* + - * 1. 触发条件：

碰撞检测：若在获取到用户按下指定按钮的瞬间，音符物体的碰撞体与得分判定区的碰撞体相交，则触发成功，根据当前音符物体的tag判定其音符类型，再加上对应的分数；若碰撞体不相交，则不进入触发成功的相应判断。

胜负判断：



如图所示，根据最终游戏得分，进行胜负与等级判断：若游戏胜利，即达到符合胜利条件的最低分，则再次进入等级的判断——胜利分段、中等分段、高分段；若未达到符合胜利条件的分数，则为低分段，游戏失败。

* + 1. 拼图游戏：
       1. 设计：
          1. 素材处理：将整张拼图分割成3\*3共9张地图碎片，以便对其进行分别控制；
          2. 生成规则：玩家共需要解谜两块拼图，分别均由9块碎片组成；9块拼图碎片在游戏开始时随机散落在一块3\*3的区域内；
          3. 触发条件：将玩家前后点击的两张地图碎片交换位置；
          4. 胜负判断：玩家成功将两块拼图按正确顺序拼出，则游戏胜利。否则游戏无法向前推进。
       2. 实现：
          1. 素材处理：确定拼图分辨率，在ps中分别将拼图的长、宽进行3等分切割，共切割成3\*3共9张尺寸相同的拼图碎片；
          2. 生成规则：

相关数据结构：

碎片列表(字段名：puzzleUnits<GameObject>)：共 9 个元素，每个元素表示一个地图碎片

位置列表(字段名：unitPosList<Vector2>): 存放一个 3 \* 3 矩阵，共9个元素，每个元素表示一个二维坐标值。

代码流程：

生成碎片列表: puzzleUnits<GameObject>

生成位置列表: unitPosList<Vector2>

对puzzleUnits进行遍历，每次遍历中，从unitPosList中随机选取一个元素，将当前遍历的碎片在生成的随机位置上实例化，再将该位置元素从列表中移除，以免遍历不同碎片时取到同一位置元素。当遍历结束时，全部碎片均已取到其唯一对应的位置，并在该位置完成实例化。至此，拼图初始化完毕。

* + - * 1. 触发条件：

相关数据结构：

碎片-位置字典(字段名：unitDic<int, Vector2>)：该字典key: (int) id, 表示该碎片的id号；value: (Vector2) position，表示对应碎片的当前位置坐标

代码流程：当用户连续选择了两块不同碎片时，对已选的两块碎片进行位置交换；位置交换后，两块碎片在 unitDic 中所对应的记录即被更新；

* + - * 1. 胜负判断：

相关数据结构：unitDic<int, Vector2> 、unitPosList<Vector2>

代码流程：

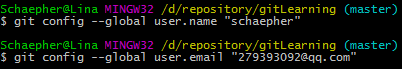
胜负条件：当每块碎片的当前位置与其对应的正确位置相等时，即判断胜利；否则未成功(检测到任意一块碎片不满足该条件即表示尚未成功)。

获取碎片的当前位置：通过该碎片的 id 号从字典中获取: unitDic<id>

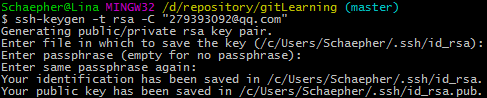
获取碎片的正确位置：id 号为 x 的碎片的正确位置即 unitPosList 下标为 x 的元素

判断：如果 unitDic<id> == unitPosList<id>，则该碎片正处于正确位置，可进行下一轮遍历，否则游戏尚未胜利。

* 1. 版本控制：
     1. 设计：采用Git作为版本控制工具
     2. 实现：
        1. 下载、安装Git for Wind
        2. 本地Git与Github的连接：
           1. 注册Github帐号；
           2. 本地配置用户名和邮箱，如图：

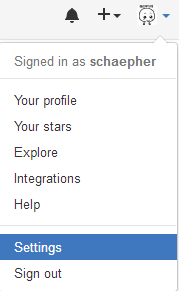


* + - 1. 生成ssh key：

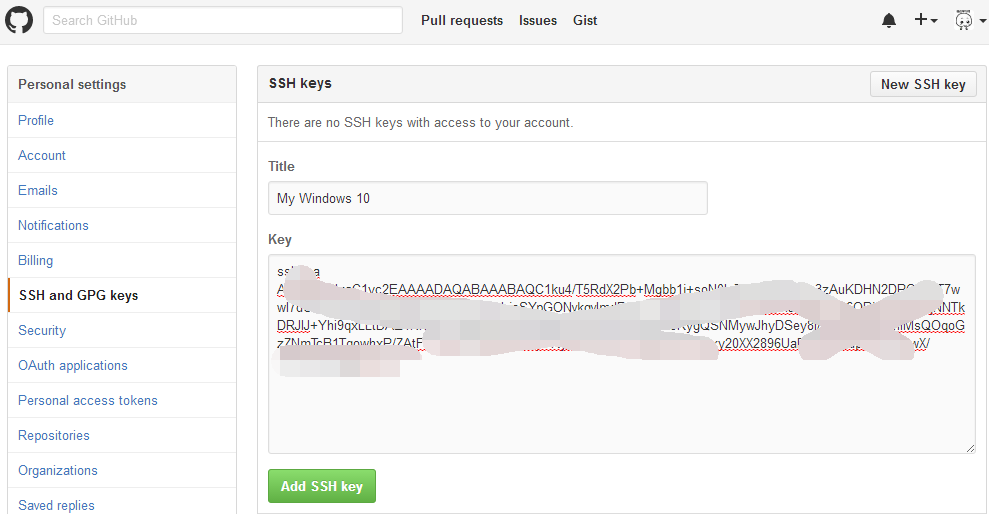


将生成的ssh key复制到剪贴板，执行 clip < ~/.ssh/id\_rsa.pub （或者到上图提示的路径里去打开文件并复制）

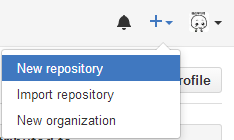
* + - 1. 将ssh key 添加至Github服务器：
         1. 打开Github，进入Settings：

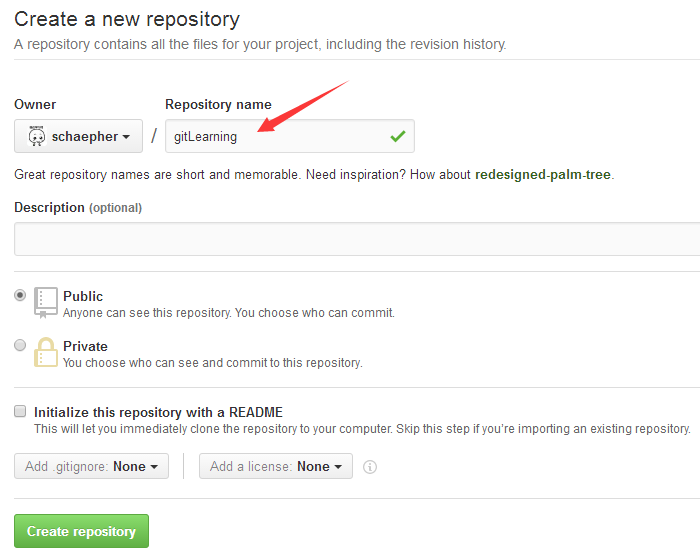
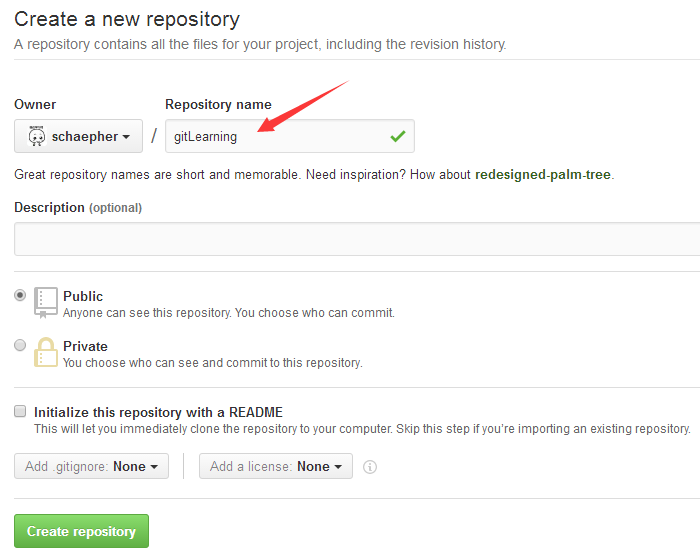


* + - * 1. 点击左边的 SSH and GPG keys ，将ssh key粘贴到右边的Key里面，点击Add SSH key。

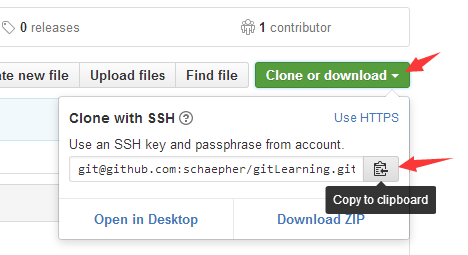


* + - 1. 创建远程仓库：





* + - 1. 关联本地仓库：

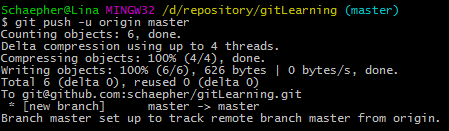


运行 git remote add origin 你复制的地址，则关联已完成 ：

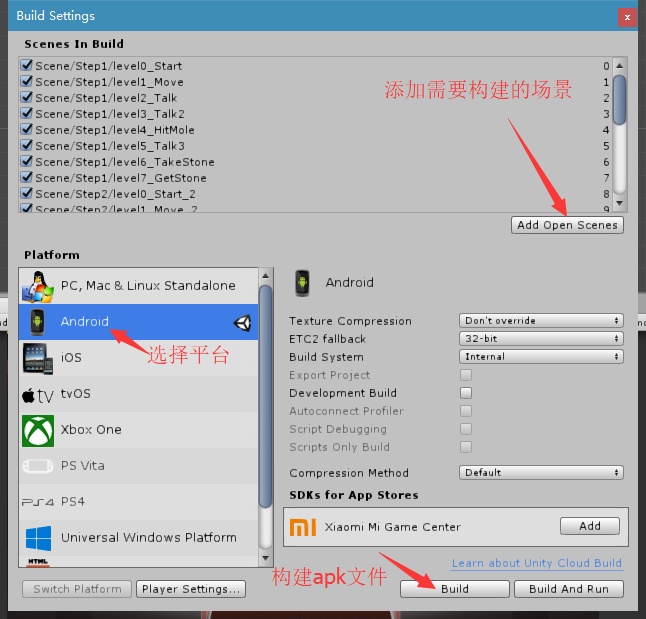
4

* + - 1. 将项目文件添加至本地仓库文件夹；
      2. 提交仓库更新内容：

执行 git push -u origin master 将本地仓库上传至Github的仓库



* + - 1. 每次对项目内容进行修改后，将记录添加并提交，即可对项目版本进行便捷的管理。
  1. 打包与安装：
     1. 设计：
        1. 构建打包：在Unity内即可完成对apk文件的构建；
        2. 安装：利用adb工具进行apk在移动设备上的安装。
     2. 实现：
        1. 打包：



* + - 1. 安装：
         1. 网络设置：确保pc与移动设备同处一个子网；
         2. 连接设备：用usb线连接pc与移动设备，并开启开发者模式；
         3. 打开cmd，设置adb：



* + - * 1. 通过WLAN模式连接设备：



* + - * 1. 若要转回usb模式：



* + - * 1. 若在WLAN模式下连接成功后，可拔掉usb连接线：



* + - * 1. 安装apk至移动设备：

