上海交通大學

数字图像处理技术与应用

结题报告

项目名称: 俄罗斯方块 AI 的机器人实现

姓　　名: 陈思哲

学　　号: 516021910038

班　　级: F1603203

同组同学: 张沛东，刘启明

联系电话: 13262292070

电子邮箱: 729020210@qq.com

2018-2019 学年第 2 学期

2019 年 6 月 8 日

目录

研究内容综述

研究任务

通过摄像头的视觉信息，经过图像处理和 AI 决策，控制机械臂进行完成俄罗斯方块。

实验要求

将摄像头采集的图像通过信息提取得到当前游戏的状态信息。实际观察发现，游戏界面左上角的白色边界框与俄罗斯方块的各位置之间的距离是固定的，而白色框与黑色背景对比明显，比较容易检测，这就为我们提取游戏状态信息提供了很好的实现途径。

得到游戏状态信息之后，还需要利用人工智能等技术，利用当前信息做出决策，这是我们控制的关键一环。实际实现中，我们参考了 Pierre Dellacherie 算法，并根据我们的游戏环境进行了参数调整和改进。摄像头获取图像之后传入算法做出决策，通过决策控制机械臂的动作。

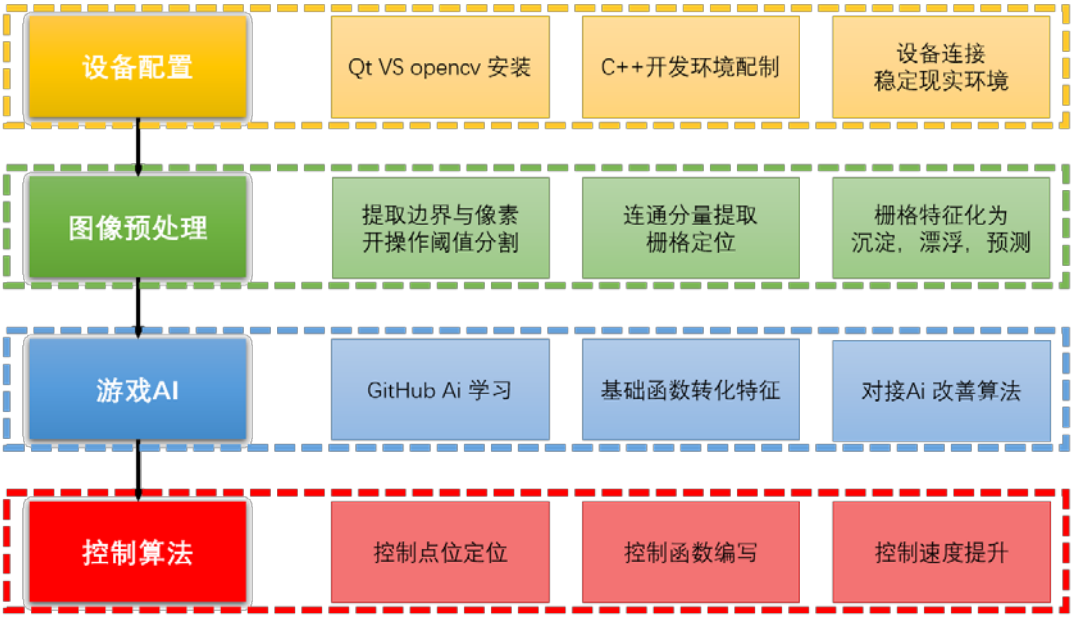
在得到 AI 给出的控制策略后，机械臂能够根据策略做出正确的反应，这一部分更多涉及到与硬件的交互。虽然对机械臂的控制可以调用相关函数实现，但是在实际测试中我们发现机械臂的动作存在很多意想不到的问题，比如下落距离不一致，延时函数失效等等。我们通过调整触控笔、延时时间等方法解决了大部分问题，使得游戏能够顺利进行。但是在少数情况下，还是会出现触控笔误触、未接触、触笔下落不到位的问题，这会对我们游戏的分数造成干扰，我们已经通过各种可能的方法将影响降至最低。

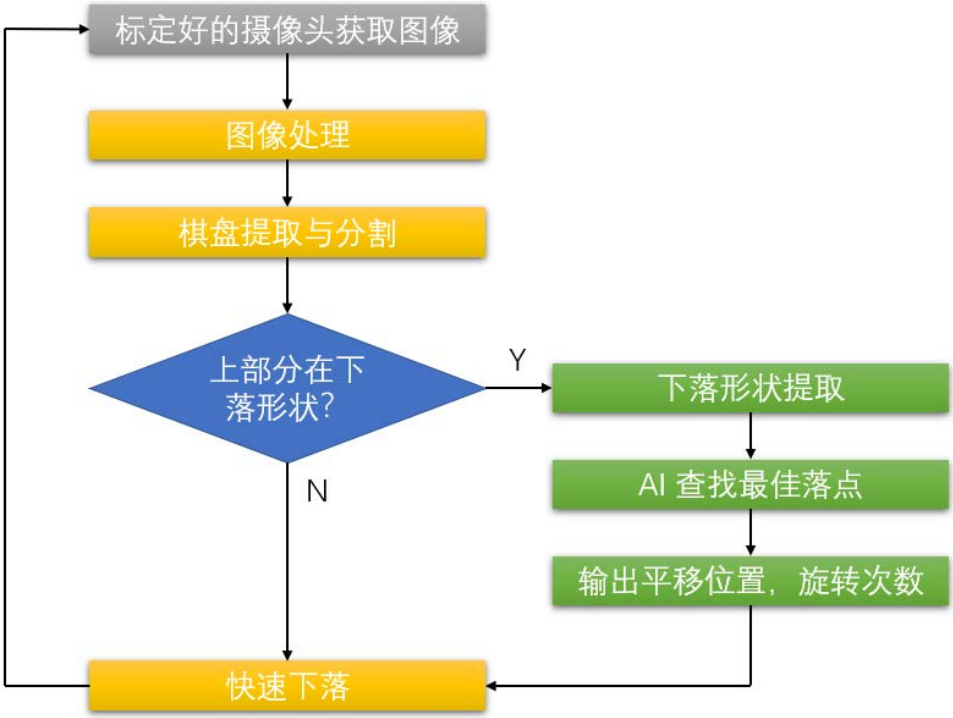
设备说明

图 1 CyberDIP 运行情况示例

本工程是依赖 C++ 11 标准编写的 CyberDIP 在 Windows 环境下的配套软件。CyberDIP 是通过计算机 USB 控制的触摸屏点击设备(中国实用新型专利 2016201772460)，通过搭载 grbl 0.8c/0.9j 的 Arduino Nano(ATmega328)控制器，CyberDIP 可以将 USB 串口发来的指令翻译成相应的二维运动与点击操作，模拟单指对屏幕的操作。 结合图像处理算法，CyberDIP 可以实现触屏手机上游戏的自动攻略功能。

项目架构





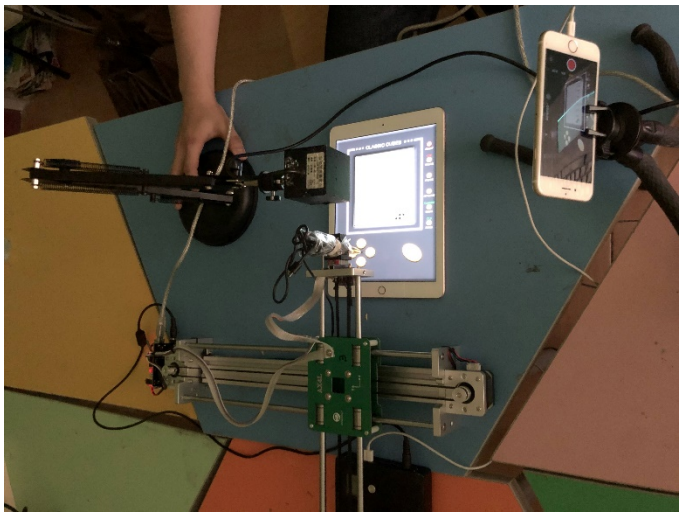
环境配置

Windows 10
Visual Studio 2017
Qt 5.11.3
OpenCV 3.4.1

研究方案

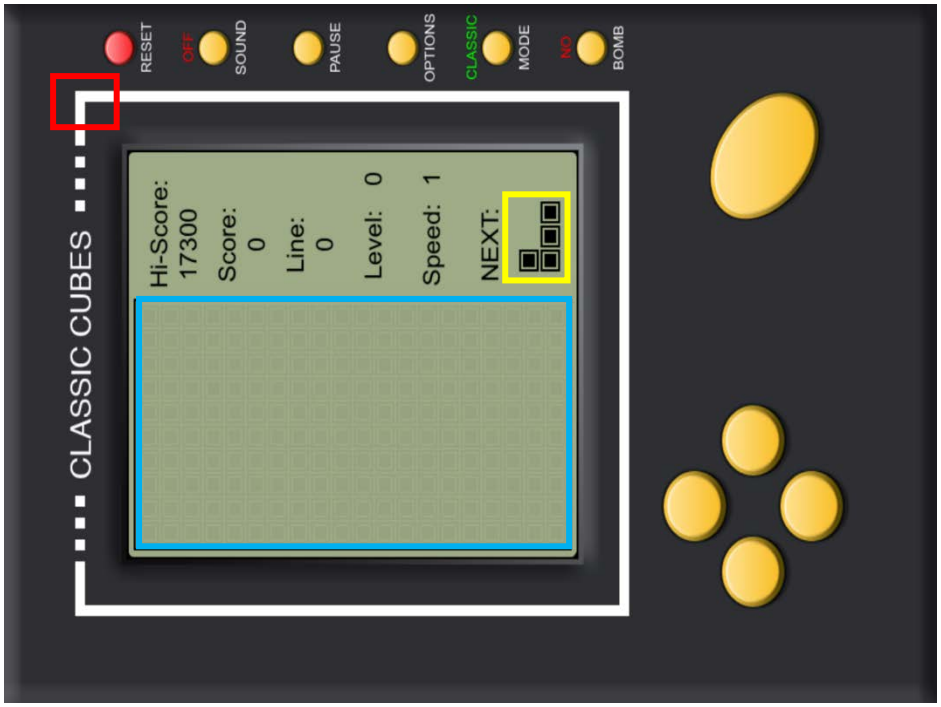
图像预处理

首先，iPad 反光严重，稍微有一些亮光，就有大面积反光。同时，每一次调试，摄像头和 iPad 的相对位置要保持固定。为了解决以上两个问题，我们将 iPad 放置于合适大小的椅子中，将摄像头固定于椅子上方，四面用卡纸包围，保证箱内为一大小位置固定的暗室，方便图像处理。首先，我们将椅子的上方用打火机烧破，用以穿过摄像头的线。剪掉椅子角的一部分，使得机械臂可以自由活动。其次，机械臂控制触控笔触控时，由于触控笔与地面没有形成闭合回路，会无法触控，需要将数据线缠绕在笔上，并垂落在桌面。另外，固定笔的螺丝会遮挡棋盘，需要拆除，用胶布与餐巾纸包裹固定笔。最后，触控笔下降的高度深浅不一，需要将笔固定在合适高度，保证深浅触控，均被 iPad 识别为一次触控。



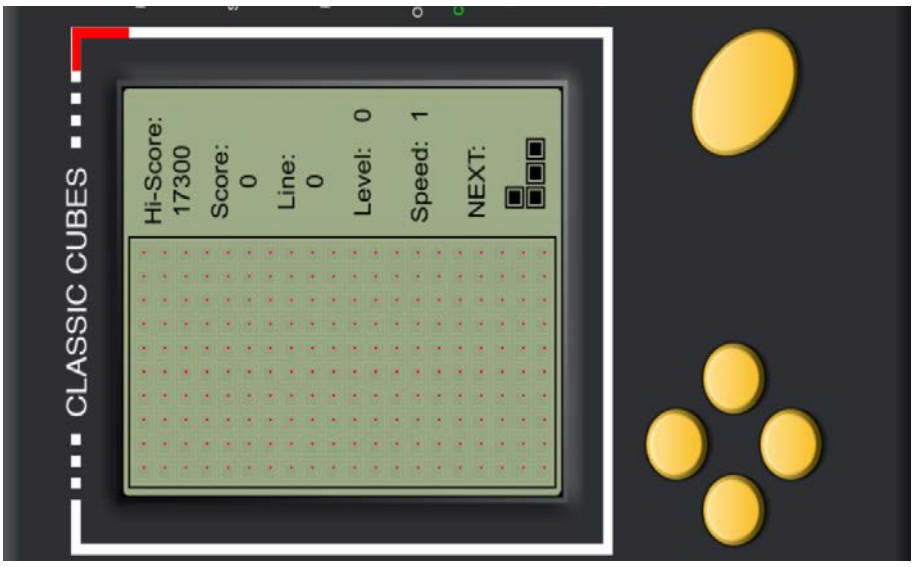
但是实际上，我们发现摄像头无法稳定地固定在椅子上方，同时暗室内的调整十分麻烦。实际测试中，我们用架子固定摄像头，关灯测试，这就需要每一次动态调整。

接下来介绍图像处理与信息提取方法，图像处理与游戏状态提取是游戏处理的关键一环，我们图像处理的基本思想是找到标定物，并利用标定物和俄罗斯方块格子之间的确定位置关系判断每一个格子是否有方块存在。游戏界面如下图所示。游戏环境高 20 个格子、宽 10 个格子，在上一个图形下落之后，上方会继续弹出下一个图形。当下方某一行被方块填满时，这一行就会被消去。如果方块达到了天花板处，游戏即结束。



观察这张图片的特点，可以发现游戏界面的背景颜色是深灰色，游戏区域与操作区域被一个很粗的白色方框隔离开。我们最需要关注的就是图 1 中蓝色方框内部每一个方格中是否有方块，如果没有方块，相应的格子里就呈现出浅灰色；如果有方块，相应的格子中便会是黑色（如图 1 中黄色方框所示）。

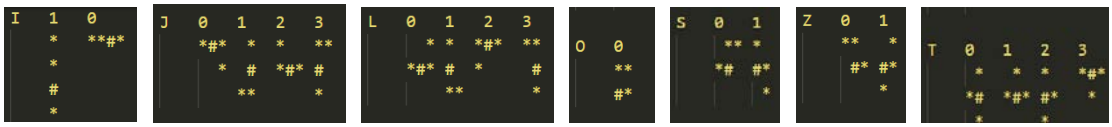
我们采用了基于标定物的信息提取方法。从图 1 中可以观察到，图片的左上方有一个明显的白色直角（如图 1 中红色方框所示），与背景黑色对比十分明显，找到它很容易。同时，这个白色直角与蓝色方框的相对位置是固定的。依据前文所说，相应位置的深浅意味着方块的存在与否，我们只需要扫描找到位置上的灰度值，即可知道该位置是否存在方块。用上述方法，我们免去了游戏中标定摄像头位置的烦恼，只需要在每次游戏开始之前，固定摄像头高度并调整屏幕水平即可。



程序实现上，我们仅仅搜索图像左上角的一小块区域，当发现位于最左上角的白色像素点时停止搜索，即可得到标定点的位置。标定点得到之后，再利用事先测量好的距离间隔得到每一个方块位置的坐标，将我们标定的点位用红色投影在输出上，调整摄像头的位置使所有格子被正确检测。我们尝试过使用四个角的标定方法，但是实际调整时，无法仅调整一个角的位置，或者固定一个角的位置调整，所以还是选择这种自适应的方法。

当一个方格中存在方块时，此方格是黑色的，否则是白色的。我们将原始图像进行二值化处理，利用上面得到的位置信息探测对应位置像素灰度值，即可知道方格存在与否。但是，游戏中的方格并不严格为一个完整黑色块，而是一个黑色块嵌套一个细的黑色边框（黄色方框中存在四个这样的方块），在探测像素值时，很有可能探测点刚好落在中间的白色区域内，这会给后续的策略判断造成极大困难。为了解决这个问题，我们对整幅图像进行了开运算，将中间细白色方框在运算中消去，这样得到的二值图即为我们所需的。定义一个长度为 201 的字符串，当相应位置有方块时设置为 1，否则设置为 0，即可用这一个字符串表示当前的状态信息。

游戏 AI



AI 方面，我们使用的是经典的 Pierre Dellacherie 算法。该算法只考虑当前，不对未来的情况进行计算。每次生成一个方块，便穷举该方块所有旋转的所有落点。一种方块最多有 4 种旋转，并且由于游戏界面是 10*20 的，故需要考虑的情况不多。算法的核心是一个评估函数。对穷举出的每一种下落情况，计算 6 个参数值，用评估函数加权求和得到一个值，该值最大的情况便是目前方块的最优下落位置。6 个参数分别是：

Landing height 表示方块下落后高度的参数。很明显，方块下落后越靠下越安全，使整体的方块堆越矮越好。Eroded piece cells 表示当前方块下落后，消除的行数和此方块中参与到行消除的方块数目的乘积，体现的是方块对消除的贡献程度。Row transitions 表示每一行从左到右扫描，方块有或无的变化次数之和。从有/无方块的格子到相邻的无/有方块格子计一次变换。其中，边框算作有方块。Colum transitions 表示每一列从上到下扫描，方块有或无的变化次数之和。从有/无方块的格子到相邻的无/有方块格子计一次变换。其中，边框算作有方块。Number of Holes 表示空洞的数量。空洞指的是，每列中某个方块下面没有方块的空白位置，该空白可能由 1 个单位或多个单位组成，但只要没有被方块隔断，都只算一个空洞。空洞的计算以列为单位。Well sum 表示井的和。井指的是某一列中，两边都有方块的连续空格，边框算作有方块。若一个井由 1 个方块组成，则为 1；若由连续 3 个组成，则和为 1+2+3 的连加。如果一个井的顶部有方块的存在，则该“井”同时也是“空洞”。

我们一直很希望利用 Python 完成项目，因为 Python 书写简单清晰，而且资源更加丰富。我们搜索到的 AI 大部分都是用 Python 写的，但是控制程序全部都

是 C++。最终，我们成功找到了 C++运行 Python 脚本的方法。使用 Python.h 库，初始化，打开 python 文件，将每个函数封装成一个对象，给输入，得到返回值，同时不能关闭文件。

我们完成了 C++调 Python 的环境配置，所以能找 AI 的范围大了许多。找到 Python AI 实现后，我们首先将 AI 从 pygame 嵌套中剥离出来，再将内置彩色的棋盘储存改为 bool 型，极大加快了运算速度。我们的测试分为线上模拟和线下实测。在线上模拟中，系统随机生成 7 种方块，AI 输出最佳落点，直接将方块移动到目标点位，快速模拟游戏现场。

实践证明 Pierre Dellacherie 算法是一个相当优秀的看一步的贪婪算法，所提出的相关的六个指标都相当成熟。但考虑到本次项目存在的主要误差在于机械误差，具体来说是随着游戏进行和俄罗斯方块海拔提高所导致的机械误操作带来的误差将被指数级放大从而破坏游戏平衡，以及算法仿真游戏与实际游戏之间拟合度的失衡。因此该项目的 AI 应该是一个更为激进的 AI，它应该更关注俄罗斯方块整体海拔的降低和如何更快消去一行。我们在 pygame 上实现了这个算法并设立了新的分数评估指标以进行算法的调参，最终得到了更为适合本项目的参数。50 * lh + 40 * epcm - 32 * brt - 98 * bct - 79 * bbh - 34 * bw，在模拟中能消除 5000 行，相比于原来的 800 行有了较大提高。

在 AI 内部逻辑上，我们发现了模拟与现实游戏间的重大偏差。每一个方块都有一个旋转中心，当我们点击旋转时，方块按旋转中心作顺时针旋转。对于 S 型和 Z 型方块，表明上看只有两种形态，但是方块从水平转为竖直时，因旋转中心的不同，可能造成一个水平的偏移。也就是说旋转一次和旋转三次，在水平方向上的位置是不同的。实际操作中，我们并不需要旋转三次，只有不旋转和旋转两种情况。所以只需要将游戏里方块的旋转中心与实际的设为一致即可。另外，AI 中纵向为 x 轴，不统一这点会对控制策略的旋转次数产生影响。

AI 判定落点合法的方法是，只要落点格子内都没有方块，就算合法。但是有可能出现，落点格子内无方块，但其上方有方块遮挡的情况，实际上为不合法。所以需要修改 conflict 判定条件。

在 AI 的封装上，改变 AI 接口，使其输入棋盘与下落方块形状，输出最佳落点及姿态。我们得到 C++传来的 200 维字符串，将其转换为 bool 棋盘信息。通过观察方块出现的规律，抓取方块形状。然后结合方块的固定出现点，对比最佳落点，得出控制序列，将其封装为 8 位字符串（1 向左，2 向右，3 旋转，0 补全 8 位）传给 C++。若没有抓取到形状，返回 8 个 0.

在线下测试中，我们发现问题并没有那么简单。游戏的速度在消除 64 行时会增加一档，同时越往后，长条 L 型方块出现的概率会越小。AI 倾向于在边上留下 L 型空隙，如果长时间不出现 L 型，那么 AI 选择用其他方块来弥补空格，导致上方遮挡了下方的长条空隙，下方再也无法消除。1988 年俄罗斯数学家证明了如果只出现 S 和 Z 型方块，游戏必然死亡。我们玩的这一版中，开发者可能也采用了逐渐降低简单的 O 或 L 型出现的概率。

在模拟中，各方块概率相同。我们设置长条形概率逐渐减少，再调整一版参数。另外，我们发现这版游戏里有炸弹，可以消除一定数量的方块，原计划对炸弹的识别与判定再写一些算法，但这样有违游戏规则，就没有实施。

控制算法

C++得到控制指令后，依次执行控制动作。若指令为 8 个 0，或依次执行控制指令后 1.5s 内，不执行控制，避免重复控制。值得一提的是，硬件自带的 delay 函数无法实现 delay。Time 模块的延迟方法在实践中也被证明无效。我们使用在函数里不停计数的方法，来实现延迟。但是 C++会将无意义的计数函数优化掉，所以我们还要不停输出计数值，并在输出结束后清除命令行输出。另外，硬件每次下落触屏的深度不一致，导致笔有时下落过深，多次触控。我们将笔固定在合适高度，保证深浅下落时都触控一次。而且，硬件多次控制后，会有偏移的机械误差，控制太久后，程序内存占用会逐步增加至 3G 以上。总之，硬件的问题，非常麻烦。

为了加快游戏速度，我们决定在完成已有操作之后让机械臂去按俄罗斯方块游戏的下键以快速下落方块，这样的策略固然会使前面的游戏进展节奏加快，但当游戏中俄罗斯方块的高度积累到相当的海拔时，这样的操作将极大可能产生误操作——上一块方块已经落地，输入的下键指令对应到下一个方块上，由此导致下一个方块落出前三行，导致无法识别最后失败。同时我们注意到随着游戏的进行，该款游戏会自动且连续地提升速度。因此经过简单决策之后我们决定采用这样的思想：当游戏时间不超过一个阈值且目前俄罗斯方块的海拔不超过某个高度时，我们会自动给机械臂输入按下键的指令，当任一条件不满足时则不按下键。

实测中我们发现，单位时间内消除的行数与总消除行数行数存在矛盾，也就是速度与鲁棒性的矛盾。我们可以在 45 分钟内消除 100 行，但是想突破记录，就得不按下键，以保持在高建筑高度时的准确控制。最终我们可以从最低难度开始，不按下键，在 100 分钟左右，消除 226 行。

研究成果

在图像处理方面，能够将摄像头采集的图像通过信息提取得到当前游戏的状态信息。具体而言，能在准确标定的前提下，切割出屏幕中的游戏区域，并且通过图像处理技术得到俄罗斯方块的棋盘状态，为游戏决策作参考。

通过 C++调 Python 的 AI，在输入棋盘信息后，自动计算出最佳落点，并给出控制动作序列。

根据序列操作，控制机械臂以合适的速度，按顺序点击相应的控制按钮。

最终整个系统需要通过摄像头不断采集数据，计算机对采集得到的图像进行处理和信息提取，并利用当前的状态信息判断下一步的最优动作，通过函数控制机械臂的动作，从而点击屏幕，完成 100 行（45 分钟）以上的俄罗斯方块消除。最终我们可以从最低难度开始，不按下键，在 100 分钟左右，消除 226 行。

研究结论

发现的问题

我们发现所给的工程代码文件在运行时即使连续点击同样的 hit 按钮，机械臂的下降高度会出现完全随机性质的降低，且这样的降低出现时间是不可控的，初步猜测是由于硬件程序问题（相关软件代码我们已经排查过），这虽然不会对前五十行的积分造成影响，但如果目标放到两百行以后，这样随机出现的误差可能会导致整个游戏的失败。

感想与体会

在本门课程中我们第一次系统性体会到了如何组合使用学习到的各种图像处理方法得到我们想要的标定结果，尤其重要的是如何与现有的对象结合，在标定过程中我们尝试了各类灰度处理和特征识别的相关方法，经过多次对比和现场的综合考量，最终才确定下目前的阈值分割+闭操作+角检测+连通性计算的方案，得到了比较理想的标定结果。

相关俄罗斯方块 AI 的 Pierre Dellacherie 算法用 python 不难实现，pygame 也提供了虚拟俄罗斯方块的接口供我们方便地调参。C++调用 python 也通过配置文件和代码层面上的数据结构转换与文件沟通实现了，真正困难的是如何进行相关策略的权衡。

实际上我认为整个游戏可以视为一个动态 SLAM 的过程，即首先通过摄像头对整个世界进行初定位并将地图保存在机器人内存中，得到的新方块传入机器人，机器人做出相应决策并执行相应动作，此时在机器人的内存推演中相当于实现了一个虚拟地图的变化，由于执行时无法得到地图（游戏情况）实际变化，事实上此时虚拟地图和实际地图可能会产生偏差，直到操作完成摄像头再次读入新画面，我们的机器人才会对内存中的虚拟地图实现校正，此时推演游戏情况和实际游戏情况才会统一。

最容易出现的偏差便出现在上述所说的机器人执行和推演的偏差，严格来说是执行机械上可能出现偏差而机器人却对此一无所知。事实证明相关算法在完美模拟情况下已经足够完成该项目，因此想要得到更高分则需要把相当多的时间和精力花在如何弥补这个偏差之上，这对我们也是一个极大考验。

意见与建议

希望课程组可以进一步优化代码，解决 comHitOnce（准确说是 comHitDown）所存在的下降高度可能变化的问题。希望课程组可以进一步优化代码，解决在运行过程中内存消耗不断升高的 bug（可能是一些应该释放掉的视频流并未释放掉）。我们诚挚建议，既然项目是大家花共同时间共同完成，课程报告一组一份，大家共同书写，打磨文字，就足以展示我们的工作。