农场游戏

作业3
第一学期，2022年
CSSE1001/CSSE7030

截止日期：2023年5月26日 16:00 GMT+10

1 简介

在作业3中，你将为*农场游戏*实现一个图形用户界面（GUI）。*农场游戏*是一个角色扮演的视频游戏，玩家在游戏中经营一个农场，种植和收获农作物，耕种和卸载土壤，购买和销售种子和产品。图1显示了最终完成的游戏的一个例子。

图1：已完成的*农场游戏*实施的屏幕截图示例。请注意，你的显示器可能看起来略有不同，这取决于你的操作系统。

与之前通过调用输入处理用户互动的作业不同，作业3中的用户互动将通过按键和鼠标点击进行。

你的解决方案将遵循讲座中提到的苹果MVC设计模式。为了帮助你实现，我们提供了模型类以及一些额外的支持代码和常量；更多细节见第4节。你需要实现一系列的视图类和控制器类。

2 设置

除了下载并解压a3.zip，要开始这项作业，你还需要通过pip安装Pillow库。关于如何安装Pillow的说明可以在这里找到[1].

3 技巧和提示

在整个编码过程中，你应该**定期**进行测试。手动测试你的GUI并定期上传到Gradescope，以确保你实现的组件通过Grade- scope测试。请注意，如果你的实现是错误的，Gradescope测试可能会在视觉上看起来正确的实现上失败。你必须根据第6节的实施细节来实现你的解决方案。请注意，你的程序中的细微差别（例如小部件大小的几个像素差异）可能不会导致测试失败。你有责任尽早并经常上传到Gradescope，以确保你的方案通过测试。

本文件概述了你的作业中需要的类和方法。我们*强烈建议*你创建自己的辅助方法，以减少代码的重复，并使你的代码更容易阅读。

除非有特别说明，你只需要做足够的错误处理，使常规游戏不会导致你的程序崩溃或错误。如果一个功能的尝试导致你的程序崩溃或表现得难以测试其他功能，而你的评分员又没有修改你的代码，请在提交作业前将其注释掉。如果你的解决方案中包含有阻止其运行的代码，你将得到0分。

你**不能添加任何进口产品**；这样做会导致**你的分数被**扣掉**最多100%**。

你**只能在本学期**使用本课程教学人员提供的任何代码。这包括支持文件中的任何代码或**本学期**以前作业**的**样本解决方案，以及课程工作人员提供给你的任何讲座或教程代码。然而，你有责任确保这些代码的风格是适当的，并且是对你正在解决的问题的适当和正确的方法。

关于tkinter的其他帮助，你可以在effbot上找到文档[2]和新墨西哥理工学院[3].

[1][2]
https://pillow.readthedocs.io/en/stable/installation.htmlhttps://web.archive.org/web/20171112065310/http://effbot.org/tkinterbook[3]
https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html

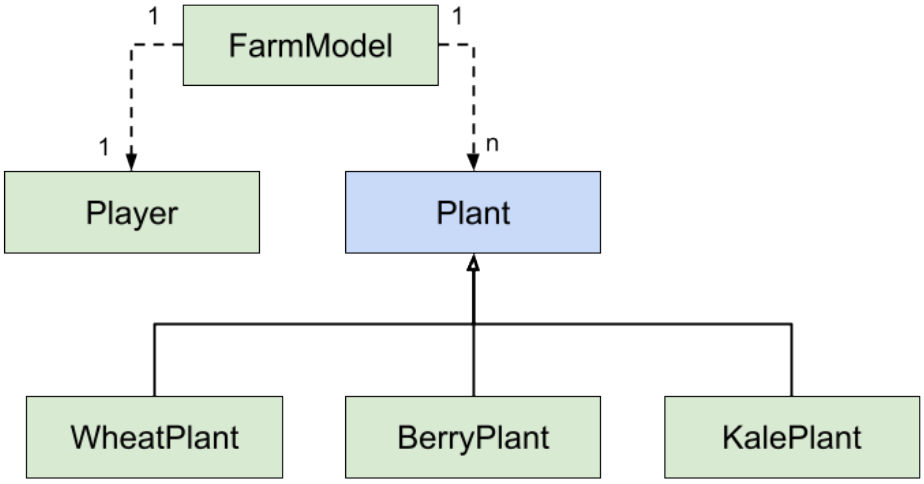

图2：model.py的类图。

4 提供的代码

本节提供了a3.zip中为你提供的文件的简要、高层次的概述。有关进一步的信息，请参见每个文件中的文档。

4.1 model.py

model.py文件提供了植物和玩家的建模类，以及游戏的整体模型。图2展示了该文件中提供的类的类图。你只需要在你的代码中实例化FarmModel就可以了。然而，你仍将通过FarmModel实例与其他类的实例进行交互。

4.2 a3_support.py

a3 support.py文件包含支持代码，以协助你编写解决方案。特别是，这个文件提供了以下内容：

1. get_plant_image_name(plant: 'Plant') -> str: 一个函数，帮助在Plant对象和其对应的图像文件名之间进行映射。

2. get_image(image_name: str, size, cache=None) -> Image: 一个根据图像文件的名称来创建、重新调整大小，并可选择缓存图像的函数。返回图像对象，它可以被渲染到一个tkinter Canvas上。***注意：*****你需要保留对所有图像的引用，或者对缓存的引用。**一旦对图像的所有引用消失，Tkinter就会删除该图像。***还要注意：*****在创建图像时必须使用这个函数。**

3. read_map(map_file: str) -> list[str] ：将一个地图文件读成一个字符串的列表，每个字符串代表地图的一个行。第一项代表最上面的一行，最后一项代表最下面的一行。

4. AbstractGrid：AbstractGrid 是一个抽象的视图类，它继承自 tk.Canvas 并为多个视图类提供基础功能。一个AbstractGrid可以被认为是一个具有设定行数和列数的网格，它支持在特定的（行、列）位置创建文本和形状。请注意，行的数量可能与列的数量不同，并且在构建AbstractGrid后可能会发生变化。

4.3 constants.py

constants.py文件包含了一组常量，这些常量在实现你的任务时将是非常有用的。如果本文件中没有指定某个细节（例如，部件的尺寸或颜色等），请参考 constants.py。

5 建议采取的方法

与早期的作业相比，你将按顺序完成任务表，开发GUI程序往往需要你在各种交互类上平行工作。你可能会发现，与其按照列出的顺序开发每个类，不如一次开发一个*功能*，并在继续开发之前对其进行彻底的测试。每个功能都需要对控制器进行更新/扩展，并有可能对一个或多个视图类进行添加。推荐的功能顺序（在读完本文档第6部分后）如下：

1. 玩游戏，主，和标题：创建窗口，确保它在程序运行时显示，并设置其标题。
2. 标题横幅：渲染窗口顶部的标题横幅。
3. 信息面板和第二天按钮（包括能够在信息面板上增加日期）。
4. 农场视角：
    - 显示文件中的基本地图。
    - 显示和移动播放器。
    - 耕种和解除耕种的土壤。
5. 项目视图：
    - 基本显示（无功能）
    - 选择项目的绑定命令。在这一点实现之后，你可以实现种植、移除植物、植物在新的一天的生长和收获。
    - 买卖物品的约束性命令。

6 实施

你必须实现三个视图类：InfoBar、FarmView和ItemView。此外，你必须实现一个控制器类--FarmGame--它将FarmModel和所有三个视图类实例化，并处理事件。

这一节描述了你所需要的实现结构，然而，它并不打算提供一个你应该处理这些任务的顺序。控制器类可能需要与视图类并行实现。请参阅第5节，了解你应该如何处理这项任务的建议顺序。

6.1 信息栏

InfoBar应该继承自AbstractGrid（见a3 support.py）。它是一个2行3列的网格，向用户显示游戏中已过的天数，以及玩家的能量和健康状况。信息条应该横跨农场和库存的整个宽度。图3显示了一个在游戏中完成的信息条的例子。你必须在这个类中实现的方法是：

- init (self, master: tk.Tk | tk.Frame) -> None: 设置此信息条为AbstractGrid，具有适当的行数和列数，以及适当的宽度和高度（见 constants.py）。
- redraw(self, day: int, money: int, energy: int) -> None：清除信息条并重新绘制，以显示所提供的日期、金钱和能量。例如，在图3中，这个方法被调用时，天=1，钱=0，能量=100。

Day:	Money:	Energy:
1	$0	100

图3：用新*FarmModel*的信息重新绘制后的信息条。

6.2 农场视角

FarmView应该继承自AbstractGrid（见a3 support.py）。FarmView是一个网格，用于显示农场地图、玩家和植物。图4中显示了一个完整的FarmView的例子。你必须在这个类中实现的方法是：

- init (self, master: tk.Tk | tk.Frame, dimensions: tuple[int, int], size: tuple[int, int], **kwargs) -> None: 将FarmView设置为具有适当尺寸和大小的AbstractGrid，并创建一个空字典的实例属性，作为图像缓存使用。
- 重新绘制(self, ground： list[str], plants： dict[tuple[int, int], 'Plant'], player position： tuple[int, int], player direction： str）-> 无：
  清除农场视图，然后（在FarmView实例上）创建地面的图像、
  然后是植物，然后是玩家。也就是说，玩家和植物应该在地面前面渲染，玩家应该在植物前面渲染。**你必须使用**a3 support.py**的**get image**函数来创建你的图像**。

6.3 项目视图

ItemView应该继承自tk.Frame。ItemView是一个框架，显示单个物品的相关信息和按钮。游戏中有6个项目（见constants.py中的ITEMS常数）。一个物品的ItemView应该包含以下部件，从左到右排列：

- 一个标签，包含物品的名称和玩家库存中的物品数量，物品的销售价格，以及物品的购买价格（如果该物品可以被购买；见constants.py的BUY PRICES）。
- 如果这个项目可以被购买，那么这个框架应该包含一个按钮，用于以列出的购买价格购买该项目。
- 一个按钮，用于以列出的出售价格出售物品（所有物品都可以出售）。

在ItemView上可以发生三种事件：左键点击物品框架或标签（表示用户想选择该物品），按下购买按钮（表示用户想购买其中的一个物品），以及按下卖出按钮。

图4：一个游戏中的FarmView例子。

按钮（表示用户想出售其中的一个物品）。这些按钮的回调必须在控制器中创建（见FarmGame）并通过构造函数传递给每个ItemView。

你必须在这个类中实现的方法是：

- init (self, master: tk.Frame, item_name: str, amount: int, select_command : Optional[Callable[[str], None]] = None, sell_command : Optional[Callable[[str], None]] = None, buy_command : Optional[Callable[[str], None]] = None) -> None ：
  设置ItemView作为一个tk.Frame运行，并创建所有的内部部件。设置了绑定买入和卖出按钮，分别用相应的项目名称调用买入命令和卖出命令。当左键点击ItemView框架或标签时，将选择命令与相应的项目名称绑定。**注意**：这三个回调的类型提示为Optional，如果你还没有实现这些回调，可以传入None（也就是说，在开发时，先传入None，等你完成其他任务后再挂上功能；见第5节）。

- update(self, amount: int, selected: bool = False) -> None：更新标签上的文字，并适当地更新这个ItemView的颜色。关于这些的详细介绍，请看图5。***注意：***你应该在这个方法中配置现有的小部件。你**不能**破坏和重新创建内部的小部件。

注意：处理回调是一项高级任务。这些回调将在控制器类中创建，因为这是你能获得所需建模信息的唯一地方。在这项任务中，首先尝试正确地渲染ItemViews，不使用回调。然后将这些视图整合到游戏中，再进行选择命令的工作。一旦可以从库存中选择物品，就可以进行种植、生长和移除植物的工作。建议你把购买和销售功能留到任务的最后。

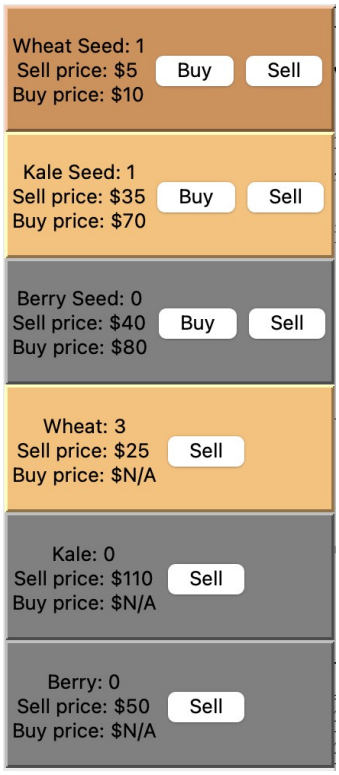

图5：整个游戏中所有6个ItemView实例的例子。注意：有些操作系统会保留按钮后面的默认背景颜色。虽然你不会因此而被标记，但如果你想设置按钮后面的颜色，你可以通过highlightbackground kwarg来实现。

6.4 农场游戏

FarmGame是整个游戏的控制器类。控制器负责创建和维护模型和视图类的实例，事件处理，以及促进模型和视图类之间的交流。图1提供了一个*FarmGame*外观的例子。某些事件应该引起如表1的行为。你必须在这个类中实现的方法是：

- init (self, master： tk.Tk, 地图文件 ： str ）-> 无：设置FarmGame。这包括以下步骤：
  - 设置窗口的标题。
  - 创建标题横幅（你必须使用获得图像）。-
  - 创建FarmModel实例。
  - 创建你的视图类的实例，并确保它们以图1所示的格式显示。
  - 创建一个按钮，使用户能够增加一天的时间，这个按钮应该有 "第二天 "的文字，并显示在其他视图类的下面。当这个按钮被按下时，模型应该推进到第二天，然后视图类应该被重新绘制以反映模型的变化。
  - 将handle keypress方法与'<KeyPress>'事件绑定。
  - 调用重绘方法以确保视图根据当前模型状态进行绘制。
- redraw(self) -> None：根据当前的模型状态重绘整个游戏。

- handle keypress(self, event: tk.Event) -> None：当一个按键事件发生时，将调用一个事件处理程序。应该按照表1触发相关行为，并导致视图更新以反映变化。如果一个键被按下但没有对应的事件，它应该被忽略。

- select item(self, item name: str) -> None：给予每个ItemView选择项目的回调。这个方法应该把选中的项目设置为项目名称，然后重新绘制视图。

- buy item(self, item name: str) -> None：给予每个ItemView的购买物品的回调。这个方法应该使玩家试图以BUY PRICES中指定的价格购买给定的物品名称的物品，然后重新绘制视图。

- sell item(self, item name: str) -> None：给予每个ItemView的出售物品的回调。这个方法应该使玩家试图以SELL PRICES中指定的价格出售给定的物品名称，然后重新绘制视图。

活动	行为
钥匙新闻界：'w'	玩家试图向上移动一格。
钥匙新闻界：'a'	玩家试图向左移动一格。
钥匙新闻界：'s'	玩家试图向下移动一格。
钥匙新闻界：'d'	玩家试图向右移动一格。
钥匙新闻界：'p'	如果一个项目被选中，并且该项目是一个种子，尝试在玩家的当前位置种植该种子。如果该位置没有土壤，该位置已经有植物存在，或者有种子 目前没有选择，什么都不做。
钥匙新闻界：'h'	试图从玩家的当前位置收获植物。如果玩家的当前位置没有植物存在，或者植物还没有准备好被收割，则什么都不做。如果收获成功，将收获的物品添加到玩家的库存中，如果该植物在收获时应被移除，则移除 农场的植物。
钥匙新闻界：'r'	试图将植物从玩家的当前位置移走。请注意，这并不是在收获植物，也不要求植物已经准备好被收获。如果没有植物存在于 玩家的当前位置，什么都不做。
钥匙新闻界：'t'	试图从玩家的当前位置耕种土壤。如果 该位置不包含未耕种的土壤，什么都不要做。
钥匙新闻界：'u'	试图从玩家的当前位置开始翻耕土壤。如果该位置没有耕种的土壤，则不做任何事情。如果 含有植物的位置，不要翻动土壤。
左键点击一个项目在库存中	如果玩家拥有该物品的数量不为零，则设置 所选项目的名称。
按钮按钮(购买按钮上项目在库存)。	试图购买该物品。
按键（在发明中的项目上卖出但是--吨）。	如果玩家拥有该物品的非零数量，则尝试出售其中一个物品。

(Tory)	

表1：事件和其相应的行为。

6.5 play_game(root：tk.Tk, 地图文件：str) -> None **function**

玩游戏的功能应该相当简短。你应该

1. 使用给定的地图文件和根tk.Tk参数构建控制器实例。
2. 确保根窗口保持开放，倾听事件（使用mainloop）。

6.6 主函数

主要的功能应该是：

1. 构建根tk.Tk实例。
2. 调用play game函数，传入新创建的根tk.Tk实例，以及你喜欢的任何地图文件的路径（例如 "maps/map1.txt"）。

7 研究生的任务：文件菜单

CSSE7030的学生需要在游戏中实现一个文件菜单，如本节所述。文件菜单应该包含一个 "退出 "按钮，它可以优雅地结束程序（类似于点击左上角的X）。它还应该包含一个 "地图选择 "选项，通过文件对话提示用户选择地图文件，然后用该地图启动一个新游戏。在新游戏开始时，玩家被设定为没有钱，能量充足，并且处于第1天。玩家应该处于（0，0）的位置。图6显示了这个文件菜单在Mac OS上的样子。在不同的操作系统上，它可能显示在不同的位置（例如，在Windows上，菜单会显示在窗口内）。图7显示了文件对话在Mac OS上的情况，图8显示了通过该功能从提供的地图中加载 "maps/map2.txt "后游戏的直接情况。

注意：您可以假设我们只用有效的地图文件来测试您的程序。但是，你不可以假设地图文件的尺寸与最初加载的地图相同。你可以在你的FarmView类中添加一个公共的清除缓存方法来帮助完成这个任务。

图6：Mac上的文件菜单（在不同的操作系统上可能看起来不同）。

8 评估和评分标准

这项作业评估了课程学习目标：

1. 应用程序结构，如变量、选择、迭代和子程序、
2. 应用基本的面向对象的概念，如类、实例和方法、

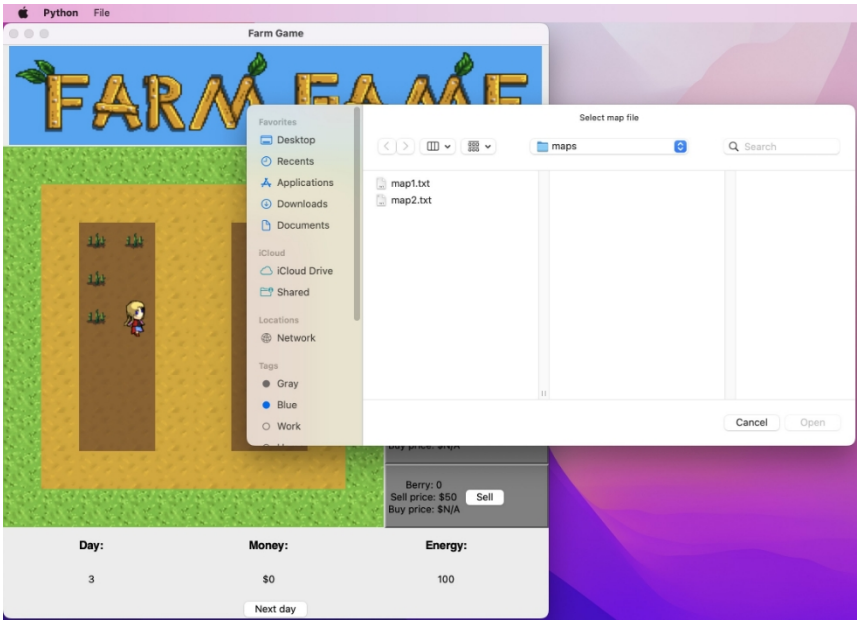

图7：导航到地图目录后提示新的地图文件（在不同的操作系统上可能看起来不同）。

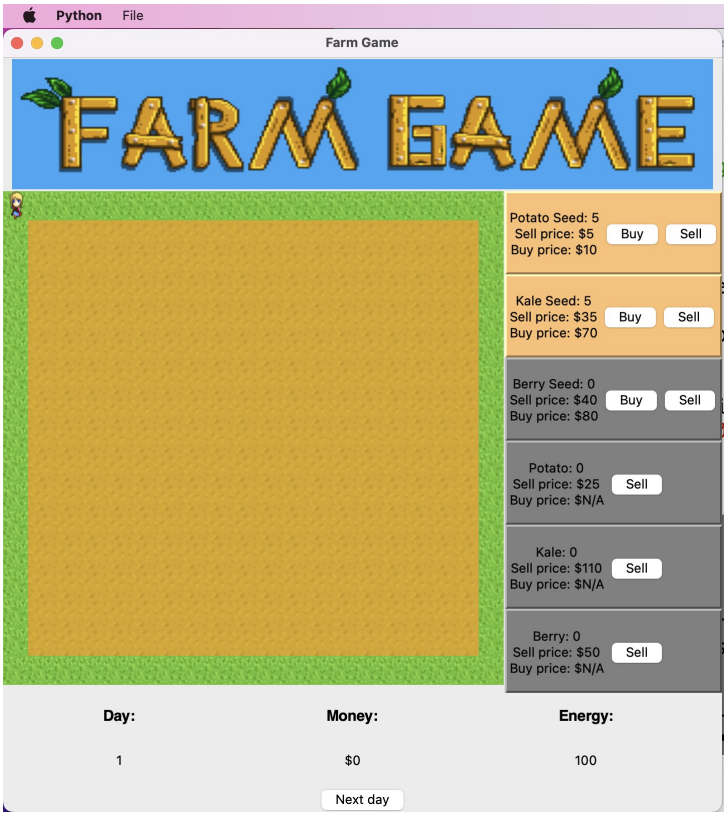

图8：通过文件菜单上的 "地图选择 "功能加载 "map/map2.txt "后的游戏。

3. 阅读和分析他人编写的代码、
4. 分析一个问题并设计一个解决问题的算法、
5. 阅读和分析设计，并能够将设计转化为一个工作程序，以及
6. 应用测试和调试的技术，以及
7. 设计和实现简单的图形用户界面。

8.1 功能标示

你的程序的功能将在70分的总分中被评分。与作业1和2一样，你的作业将通过一系列的测试，你的功能分数将与你通过的加权测试的数量成正比。在作业到期日之前，你会得到一个功能测试的 *子集*。

你的作业将被测试游戏功能的特点。自动测试将播放游戏并试图识别游戏的组件，然后测试这些组件在游戏中的功能。在提交之前，运行功能测试，以确保你的应用程序的组件可以被识别。如果自动评分器无法识别组件，**即使你的作业具有功能性，你也**不会得到分数。在提交前提供的测试将帮助你确保所有的组件都能被自动导航仪识别。

你还需要对你的程序进行 *自己的*测试，以确保它符合作业中给出的 *所有*规范。仅仅依靠所提供的测试可能会导致你的程序在某些情况下失败，你将失去一些功能分数。

你的程序必须通过Gradescope运行，它运行Python 3.11。部分解决方案将被标记，但如果你的代码中有错误，导致解释器不能执行你的程序，你的功能分数将为零。如果你的代码中有一部分导致解释器失败，请注释掉该代码，以便其余部分能够运行。你的程序必须使用 Python 3.11 解释器运行。如果它在另一个环境中运行 (例如 Python 3.9, 或 PyCharm)，而不是在 Python 3.11 解释器中运行，你的功能分数将为零。

8.2 样式标记

你的作业的风格将由导师评估。文体评分的满分是30分。

标记你的代码风格的关键考虑是代码是否容易理解。代码风格有几个方面有助于代码的易懂程度。在这项作业中，你的代码风格将根据以下标准进行评估。

- 可读性
  - 程序结构：代码的布局使其更容易阅读和遵循其逻辑。这包括使用空白来强调逻辑块。
  - 标识符名称：变量、常量、函数、类和方法的名称要清楚地描述它们在程序逻辑中所代表的内容。**不要**使用 *匈牙利符号*作为标识符。
- 文件

  - 内联注释：所有重要的代码块都应该有一个注释来解释逻辑是如何运作的。对于一个小的方法或函数，逻辑通常应该从代码和文档串中清楚地看到。对于长的或复杂的方法或函数，每个逻辑块都应该有一个行内注释来描述其逻辑。
  - 翔实的文件串：每个类、方法和函数都应该有一个总结其目的的文档串。这包括描述参数和返回值，以便其他人能够理解如何正确使用该方法或函数。

- 代码设计
  - 单一的逻辑实例：在你的程序中不应该有重复的代码块。任何需要多次使用的代码都应该作为一个方法或函数来实现。
  - 控制结构：通过对控制结构（如循环和条件语句）的良好使用，使逻辑结构简单明了。

- 面向对象的程序结构
  - 模型视图控制器：GUI的视图和控制逻辑与模型明确分开。模型信息存储在控制器中，并在需要时传递给视图。
  - 抽象：类的公共接口是简单和可重用的。使得模块化和可重复使用的组件能够抽象出GUI的细节...
  - 封闭性：类被设计成具有状态和行为的独立模块。方法只直接访问它们被调用的对象的状态。方法永远不会更新另一个对象的状态。
  - 继承性：子类扩展了他们的超类的行为，而没有重新实现行为，也没有破坏超类的行为或设计。抽象类被用来有效地分组子类之间的共享行为。

9 作业提交

你的作业必须以a3.py的形式通过Gade- scope上的作业三提交链接提交。你不应该提交任何其他文件（如地图、图片等）。你不需要重新提交任何提供的文件。

在没有批准延期的情况下逾期提交作业，将按照课程ECP的规定受到逾期处罚。在特殊情况下，你可以提交延期申请。

所有延期申请必须在提交截止日期前**至少48小时，在**昆士兰大学申请延长程序性评估表格上提交：https://my.uq.edu.au/node/218/2。