程序设计实习大作业报告

43 组

（一）程序功能介绍

我们的大作业参考并且实现了已有项目 https://ncase.me/trust/的功能，并添加了一些功能。总体上，程序主旨是研究博弈论中的囚徒困境问题，并且实现了博弈论中经典囚徒困境问题[1]的决策的可视化，同时提供了足够丰富的策略以便于研究不同策略在长期竞争中的表现，并提供了为用户提供了修改各个参数的接口，有助于加深对于囚徒困境问题及其中所演化的信任的理解。以下是详细功能的介绍：

1. 主体框架：程序总共分为 6 个页面，用户可以通过下方的 1-6 这 6 个按键跳转到任意一个页面，其中第 1 个页面为起始页面，第 2 个为我们囚徒困境模型的介绍，第 3-4 个页面提供了该模型下 one vs one，也就是双人囚徒困境的过程的可视化演示，第 5-6 个页面提供了不同策略在长期竞争下表现的可视化演示。此外，用户还可以通过点击程序右下方的“music”按钮循环播放音乐“The Road Not Taken”。

2. one vs one 动画演示：

（1） 在第 3 个页面中，程序为用户提供了最基本 one vs one 的功能实现，用户可以通过按下“cooperate”或者是“cheat”按钮来选择是否与对方合作，并且将通过动画表现双方的决策以及产生的结果，同时用户可以通过上方的分数显示框实时看到自己以及对手在多轮博弈下积累的分数。在这一页面中，我们提供了代表 5 种不同策略的小人，他们将分别与用户博弈。

（2） 在第 4 个页面中，程序为用户展示了我们为使模型更贴合现实引入的随机性，用户可以在引导下看到小人偏离自己的选择（或者策略）产生错误选择的动画。

3. 多人多策略锦标赛动画演示以及参数的自由实时调整：

在第 5 个页面中，程序提供了我们一轮多人多策略锦标赛流程演示，随后在第六个页面中提供了一个多人多策略锦标赛的模型，它实现了下面的功能：

在左边的锦标赛 ui 中：

（1） 用户可以通过按下“start”按钮开始无限循环的锦标赛过程，动画将演示在当前参数下每一轮的所有参赛者的得分，以及淘汰者退出，随后新加入者的过程。

（2） 用户可以通过按下“step”按钮进行步进操作，程序将按照：加入新人、进行一轮比赛并且显示所有人得分、淘汰的循环顺序进行动画演示，每按下一次“step”就依次展示其中的一个过程。

（3） 用户可以通过按下“reset”按钮，终止当前的锦标赛并返回到初始设置。

在右边的参数调整 ui 中：

（1） Population 部分：用户可以通过拖动滑动条或者输入数字来调整代表该策略的小人数量，同时代表其他策略的小人数量将会随机减少以保持总人数的一致。

（2） Payoffs 部分：用户可以在这里调整不同选择下双方的得分以更深入地比较不同策略。同时，无论左边的锦标赛演示在任何状态下（“start”后的无限循环状态，或者是“step”的步进状态），Payoffs 的修改都将同步到锦标赛中，并将在下一轮锦标赛中体现出来。

---

[1] 我们的基本囚徒困境模型可以如下描述：两个小人可以分别选择合作或者背叛对方，并且会根据对方以及自己的选择获得相应的分数：若双方均合作，双方均得 2 分；一方合作、另一方背叛，合作者得-1 分，背叛者得 3 分；双方均背叛，双方均得 0 分

（3） Rules 部分：在这里，用户可以对锦标赛的基本规则进行调整：包括每一轮锦标赛中，两两间的博弈次数；每一轮锦标赛中根据分数淘汰也即补充进来的人数；在比赛过程中，小人们犯错的概率。与上面的 Payoffs 部分一样，这些参数的修改都是实时的，他们将即时地体现在下一轮锦标赛中。

（二）项目各模块与类设计细节

1. 项目配置与构建模块

CMakeLists.txt:

- 定义项目名称和版本。
- 设置 C++标准为 17。
- 启用 Qt 的自动 UI、MOC 和资源文件处理。
- 查找 Qt 的 Widgets 和 Multimedia 模块。
- 包含源代码文件和资源文件。

2. 主窗口模块

MainWindow 类：继承自 QMainWindow。

- 为应用程序的主窗口，包含菜单栏、状态栏以及中心部件。
- 负责应用程序的生命周期管理以及用户交互的顶层逻辑。
- 功能实现：

(1) 页面跳转：在 MainWindow 中通过加入 stackedWidget 并将其提升为自定义的 qt 设计界面类。加入下方按钮，通过信号与槽的绑定实现页面切换。

(2) 音乐播放：在成员函数 Play 函数中实现控制

3. UI 页面模块

包含以下几个类：

(1) pg_welcome：继承自 QWidget。

Ui::pg_welcome: UI 布局类，自动生成，负责欢迎页面的布局。

(2) pg_rules：继承自 QWidget。

Ui::pg_rules: UI 布局类，自动生成，负责规则页面的布局。

qtextedit 文字插入，基本规则介绍。

(3) pg_twoplayers：继承自 QWidget。

负责展示两个玩家的交互界面，包括动画效果。

- 动画展示：ui 设计界面将上传图片与 qlabel 进行绑定，通过信号与槽的绑定实现图片的动画效果。文字引导式介绍。
- 动画实现：
  (1) 通过成员函数 on_cheatButton_clicked 和 on_cooperateButton_clicked 实现选择每轮合作或者背叛（传入对应选择参数至此轮对手对应的函数中），并开始一轮的动画实现。每进行五轮更新一次对手，用户方将面对五种策略的对手（即 copycat、cheater、cooperator、grudger、detective）。
  (2) 通过 QPropertyAnimation 类下成员函数将带有图片的 qlabel（即小人和硬币），实现点到点的位移、移动路径、时长的设置。使用 QSequentialAnimationGroup 实现多个图片不同动画片段的整合以同时进行。使用信号和槽的绑定实现不同动画先后

顺序进行。

(3) 依据每轮选择合作或背叛，传入参数至 Match_Result 类下成员函数 myresult，返回本轮双方是被合作还是被背叛，依据两者四种结果可能性给出响应（即为 reaction 函数），显示对应小人心情的图片，显示对应结果的 machine 图片。

(4) 上方计分牌，将每轮的 current_score、opponent_score、total_score 与 ui 连接，在每轮投币后更新分数。

(4) pg_mistakes：继承自 QWidget。

负责展示游戏中的失误或特殊动画的页面。

- 引导式介绍错误机制。与 pg_twoplayers 动画实现基本一致，在第二轮用户方更新错误动作，传入参数默认为背叛。上方更新文字说明。

(5) pg_tournament：继承自 QWidget。

Ui::pg_tournament: UI 布局类，负责锦标赛页面的布局。

- 引导式说明多人多策略竞标赛中，不同策略小人如何两两进行十轮博弈，并且展示各自得分。点击 continue 按钮即可演示每两个小人之间对局结果。

(6) pg_allplayers：继承自 QWidget。

Ui::pg_allplayers: UI 布局类，负责所有玩家页面的布局。

- ui 设计界面插入 tabWidget，实现切换自定义的 population、payoffs 和 rules 三界面。各界面有相应的参数值传至相应逻辑部分。

(1) 将 population 界面调整八种不同策略小人的 widget 提升为自定义的类 slider，以实现拖动滑条和数字变化显示的关联。

(2) Payoffs 界面插入 spinbox 以调整相关参数。

(3) 将 rules 界面相关参数调整的 horizontalSlider 和 spinBox 关联，以实现拖动滑条和数字变化显示的关联。

4. 基本囚徒困境模型的逻辑模块

(1) Match_Result:

- 存储单个比赛的结果，包括双方的行为和得分变化。

(2) Judge:

- 负责管理比赛逻辑，包括存储比赛结果、历史选择和判定矩阵（以及判定矩阵的修改），以及根据上面数据完成两个小人一对一博弈。
- 功能实现：
  (1) 判定矩阵修改：由成员函数 reward_reset 实现
  (2) 根据存储数据完成小人一对一博弈的选择输出：由成员函数 match 实现

5. 小人模块

(1) Player：继承自 Qwidget

- 小人的基类，定义了小人的基本属性和行为，如得分、类型，犯错概率等
- 以及小人的图片，控制其他辅助部件的动画
- 同时负责进行决策，以及随机犯错等功能的实现。

- 功能实现：
  (1) 小人决策：由纯虚函数 choice 实现，并在派生类实现具体的策略。
  (2) 随机犯错：由成员函数 random_mistake 实现。

(2) Player_Copy_Cat：继承自 Player 类
- 实现从众策略：永远复制对方上一次的决策，在最开始总是选择合作。

(3) Player_Cheater：继承自 Player 类
- 实现背叛策略：永远背叛。

(4) Player_Cooperator：继承自 Player 类
- 实现合作策略：永远合作。

(5) Player_Grudger：继承自 Player 类
- 实现记仇策略：在最开始选择合作，但是当被对方背叛后，永远背叛。

(6) Player_Detective：继承自 Player 类
- 实现侦探策略，尝试根据历史行为分析对手的决策：在最开始四次博弈中采取合作，背叛，合作，合作的决策，随后在这四次博弈中对方若有选择过背叛，则从第 5 次起采取模仿策略，反之采取背叛策略。

(7) Player_Random：继承自 Player 类
- 实现随机策略：随机选择合作或作弊。

(8) Player_Copy_Kitten：继承自 Player 类
- 实现更加宽容的模仿策略：只有当被对方连续背叛两次时，才会选择背叛对方。

(9) Player_Simpleton：继承自 Player 类
- 实现单纯策略：当对方选择合作时采取和上次一样的决策，反之采取相反的决策。

6. 辅助控件模块

slider:
自定义的滑块控件，提供一组绑定好的 QSlider 和 QSpinBox。

7. 辅助工具模块

Connection_Line:
表示玩家间关系的连接线。主要负责动态展示对局进度，具有出现、淡出和移动动画。

Trash_Can:
用于存储和管理不再需要的 Connection_Line 对象。防止重复删除以及简化接口。

8. 锦标赛管理模块

(1) Tournament_Worker：继承自 QObject
- 在后台负责运算锦标赛结果以及控制小人的 ui。

- 功能实现：

(1) 所有人加入到锦标赛数据集以及做出相应 ui 更新：在成员函数 LetThemIn 中实现。

(2) 进行一轮单循环博弈(两两之间进行多轮博弈)计算以及 ui 更新：在成员函数 Competition 中实现

(3) 淘汰分数最低的若干人并做出相应 ui 更新：在成员函数 KickThemOut 中实现

(4) 集成控制 ui 给出指令并作出相应控制：集成在成员函数 Tournament_Round(负责实现“start”控制功能)，Work_OnStep(负责实现“step”(步进)功能)中，通过调用(1)-(3)中的模块实现控制，在成员函数 reset 实现“reset”功能。

(5) 在一轮锦标赛后从 Tournament 中读取更新后的参数数据：通过发送信号 Update_signal 并在 Tournament 的成员函数 Update 中实现。

(2) Tournament: 继承自 QWidget

- 接收控制 ui 给出的锦标赛进程的信号，并将信号集成传入 Tournament_Worker 中以控制锦标赛进程，同时更新部分 ui。
- 接收参数 ui 给出的参数调整的信号，并根据当前情况做出相应 ui 响应，并将调整后的参数读入缓存，在 Tournament_Worker 发出更新信号后将缓存中的数据输入到负责计算的 Tournament_Worker 类中。
- 功能实现：

(1) 实现按下“start“,“step”,“reset”按钮后将信号同步到 Tournament_Worker 中并实现相应锦标赛流程管理：在构造函数中将“start“,“step”被按下的信号绑定到 Tournament_Worker 类的成员函数 Button_OnPush 上，并在该函数中做相应处理。在构造函数中将“reset”被按下的信号绑定到成员函数 reset 上，实现 Tournament 以及 Tournament_Worker 的全部 ui 以及数据的重置。

(2) 实现修改不同策略小人数量后更新后即时更新所有策略小人数量的 ui，同时使(无论是否正在进行)锦标赛立刻重置为一轮锦标赛的初始状态：在构造函数中将代表小人数量的 ui 更新的信号绑定到成员函数 PlayerNum_Change 以及 Tournament_Worker 类的成员函数 Button_OnPush 上，前者负责实现所有代表小人数量的数据的更新(在保证总和一定的前提下随机修改其他策略的小人数量)以及 ui 的修改，后者负责更新 Tournament_Worker 类的计算状态。

(3) 实现修改判断矩阵和规则参数后在下一轮锦标赛前即时更新到 Tournament_Worker 中：前者的 ui 数值变化信号绑定到成员函数 ValueMatrix_Change 上，并在其中将最新数据读入缓存中，后者信号则绑定到成员函数 Rule_Change 上并完成类似功能。

9. 资源和多媒体模块

- 使用 Qt 资源系统(.qrc)来管理多媒体资源，包括图像、音频等。

（三）小组成员分工情况

- 王梓桐：整体架构；基础逻辑部分，包括 Judge，Player，Match_Result；以及 sandbox 部分的左侧显示与适配，包括细化 player 与 ConnectionLine 的图像、移动与动画，以及修改部分 Tournament 类的方法使之适配
- 张语菲：stacked_widget 的 ui 页面显示部分，pg_twoplayers、pg_mistakes、pg_tournament 动画实现，pg_allplayers 参数调整部分 ui 控件插入
- 黄天域：锦标赛管理模块的逻辑部分，该模块整体框架的搭建；小人模块部分策略以及功能的实现；音乐播放功能的实现；pg_twoplayers 的博弈逻辑。

（四）项目总结与反思

●**总结：**项目充分利用了 Qt 的平台特性，取得了较为满意的成果.

1.在逻辑上较为完美的实现了我们参考的项目 https://ncase.me/trust/，同时项目模块化程度高，各个页面、功能均由专门的文件负责实现，使得项目在需要调整、修改时能够较为快速的定位到关键。

2.充分使用 Qt 自带的强大的 GUI 设计界面完成了底层 ui 设计，减少了代码量的同时更加符合 Qt 的风格。

3.所有动画均使用 QPropertyAnimation 完成，代码简洁的同时保证了动画的流畅。

●**反思**

1.在处理较为复杂的逻辑需求时设计思路不够成熟，代码在逻辑处理上不够简明，为维护和完善带来了一定障碍。

2.UI 显示较我们参考的项目比较为粗糙，许多细节略有瑕疵，程序在处理短时间内大量的 UI 控制请求的功能上做得不够完善。

3.不足的原因总结：

(1)初期规划时对项目部分较稳复杂的逻辑需求没有十分清晰，使得项目在复杂需求部分处理较为冗长，不够简明。

(2)初上手 Qt 时，对 Qt 的底层逻辑理解不够深入，为我们的开发带来了一定的障碍，导致一些功能的实现不符合预期，同时也为定位并修正 bug 带来了一定困难。

4.一些克服困难的尝试：

(1)面对出现的各种 bug 和预期之外的行为，积极搜索和阅读相关技术博客、社区讨论和 qt 官方文档。这些资源不仅帮助理解问题的根本原因，还提供了实际可行的解决方案。通过查找和应用已有的解决方案，能够更快地克服障碍并有效地改进代码质量。

(2)采取了迭代开发的方法，多次试验不同的实现方式和调整。对于复杂的技术问题和不符合预期的结果，保持耐心。

●**未来展望**

在将来的 Qt 项目中，我们会花更多的时间在设计阶段，确保逻辑结构的大框架更加清晰，函数的设置和组织更加明确。同时学习和准备更多关于 Qt 中 UI 设计的技巧和方法，以实现更精美的用户界面。