**客户端框架、详细设计（不包括游戏画面渲染）**

负责人：胡俊杰

**1 程序描述**

客户端作为C/S架构的重要节点，是玩家进行游戏的主体，负责连接玩家和服务器端。客户端框架接收用户的操作和指令，向服务器发送用户的指令；同时接收服务器、游戏的反馈信息，对数据信息可视化，实时显示给用户。

**2 功能**

客户端的功能性需求在需求分析文档中已展示。根据其功能简要描述如下。

2.1 响应用户的操作请求，包括登录、 其他玩家、游戏内键盘操作等。

2.2 将玩家的操作编码，发送给服务器。

2.3 接收来自服务器的响应信息，并解码信息。

2.4 根据服务器的响应信息，给予用户可视化的反馈，如弹窗提示、游戏界面可视化等。

**3 其他需求**

性能需求方面，至少要求在I5-8250U，8G内存的条件下，稳定渲染游戏画面30FPS。

**4 输入和输出**

客户端与服务器的交互输入输出在“坦克大战传输结构.docx”中有介绍。

4.1 向服务器的请求

客户端向服务器的请求内容有2类：登录请求、游戏中用户操作发送。

4.1.1 登录请求：

用户将向服务器发送json格式的数据对象，传输协议为TCP。发送的json有2个字段：’id’，’begin\_game’，表明用户的ID和请求内容为登录并开始游戏。其格式为{‘id’: int, ‘begin\_game’: 1}。

4.1.2 用户的游戏操作

由于用户在键盘中操作，则客户端接收键盘信息后将已按下的按键发送给客户端。发送以json格式，使用UDP协议，有6个字段：’id’, ‘Up’, ‘Down’, ‘Left’, ’Right’, ‘fire’，分别表示用户账号ID，是否按下上、下、左、右、开火按键。格式为{‘id’: int, ‘Up’: 0|1, ‘Down’: 0|1, ‘Left’: 0|1, ‘Right’: 0|1, ‘fire’: 0|1}

4.2 服务器的响应报文

游戏服务器向客户端输入的内容有3类：登录确认信息、初始地图信息、实时更新变换的游戏状态。

4.2.1 登录确认信息：登录

登录的返回信息以字符串形式表示，若该字符串为用户的ID，则表示登录成功，否则登录失败。登录返回信息是TCP请求登录后的响应报文。

4.2.2 地图初始信息

地图初始信息尤其重要，因而使用TCP协议接收。地图的初始信息为json格式的字符串，主要包含地图的障碍物、草丛。其格式为{‘obs’:[[int, int], [int, int], …], ‘grass’: [[int, int], …]}

障碍物的表示方式为左上角的点坐标[int, int]，用整数形式表示，所有障碍物放置于列表中。草丛的表示方式与此相同。

4.2.3 地图实时状态

由于地图实时状态接收频率高，且实时性强，故而使用UDP协议接收，并以二进制数据流表示，该数据流的定义如下。

前24bytes：保留字符、获胜者ID、坦克数、子弹数、障碍物变化数、道具变化数，每个单位各4Bytes。获胜者ID标记是否仅剩下1个玩家，若是则记录获胜者ID，否则为-1。坦克数记录当前场上剩余坦克数目，并便于接下来的坦克数据读取。子弹数、障碍物数、道具数均用于接下来的数据解读。

往后28 \* 坦克数 Bytes：记录每个坦克的信息。坦克的信息长度28Bytes，格式为int, int, int, int, int, float, float，分别表示坦克id、生命值、剩余弹药数、该玩家杀敌数、坦克朝向、坦克中心在地图坐标系的横坐标x、纵坐标y。

往后15 \* 子弹数 Bytes：记录每颗子弹的信息。子弹信息长度15Bytes，格式为int, int, int, float, float，分别记录子弹发射者账号、子弹id、是否爆炸、子弹中心在地图坐标系的横坐标x、纵坐标y。

往后8 \* 障碍物变化数Bytes：记录被击打而破坏的障碍物。每个变化的障碍物占用8Bytes，格式为int, int，记录障碍物左上角的横坐标x，纵坐标y。

往后20 \* 道具变化数Bytes：记录变化的道具，即新产生或者消失的道具。每个道具格式为int, int, int, int, int，分别表示道具id、道具类型、状态、左上角坐标x、y。道具类型：0表示血包，1表示弹药箱。道具状态：0表示消失，1表示新产生。

最后是32Bytes，表示安全区的信息。格式为int, int, int, int, int, int, int, int，分别表示当前电网圈的左上角x, y，右下角x, y；缩圈目标左上角x, y，右下角x, y。

上述二进制数据流使用DataTransPack模块可以解包为python字典，解包后字典格式为{‘info’:[int \* 6], 'tanks': [[int \* 5, float \* 2], […], …], 'bulls':[[int \* 3, float \* 2], …], ‘obs’: [[int \* 2] ,…], ‘props’: [[int \* 5], …], ‘safe’:[int \* 8]}。字段info, tanks, bulls, obs, props, safe分别为数据头，坦克信息，子弹信息，变化的障碍物，变化的道具，安全区。使用该字典，可还原出当前帧与上一帧的变化信息。

**5 子系统分解和模块分解**

客户端的子系统可分解为客户端窗体、网络传输、游戏界面渲染三个模块，见“客户端包和依赖.vsdx”。实际设计中，使用Python 3语言官方自带的tkinter库实现客户端窗体部分；基于多线程类Thread和网络套接字Socket处理网络传输，包括向服务器发送消息，接收服务器消息并放置在队列中；基于第三方库PIL的游戏界面绘制。

客户端共开启5个线程。主线程为客户端窗口的事件队列驱动线程，负责循环处理窗口事件。剩下4个线程在网络模块中，负责监听接收、发送消息。对于接收的消息，将放置在线程公用队列中等待主线程提取。

**6 接口**

client\_win模块暴露的接口有2个：

* MainForm()：是MainForm类的构造函数，生成并返回一个客户端主窗体对象。
* MainForm.mainloop()：让主窗体MainForm开始事件驱动的循环历程。

client\_connector模块暴露的接口如下：

* Connector()：生成一个Connector对象，用于客户端的网络连接。
* udpStart()，tcp\_link()，tcp\_unlink()，send\_data\_tcp(dict)，send\_data\_udp(dict)，get\_tcp\_data()，get\_udp\_data()，end()：Connect类的公有方法，其功能见7.2.1节。

**7 数据结构和算法**

7.1 客户端界面数据结构（client\_win模块）

客户端界面类图见“client\_win类图.vsdx”。本模块依赖于官方库tkinter，项目其他模块client\_connect, ClientDisplay。

7.1.1 class MainForm

MainForm类是客户端程序的主窗体，继承自tkinter.Tk类，并且继承了组织安排窗体控件、事件驱动核心等功能。派生类增加的属性包括网络连接器connect，玩家账号player\_id，记录当前显示的页面cur\_page和三个页面：登录页面login\_page，主界面menu\_page，游戏页面game\_page。页面的切换方式为覆盖显示和消失隐藏。

MainForm方法如下：

toMenuPage(): 转移到menu\_page，包括使用Frame.lift()方法将该页面提至最顶层，覆盖其他层，修改当前页面标签，修改主窗口大小为合适值。

toGamePage(dict): 转移到game\_page。参数dict为从服务器接收的初始地图字典。算法包括使用Frame.lift()方法将该页面提至最顶层，修改当前页面标签，修改主窗口大小为合适值，最后调用game\_page.game\_start()，以把初始地图字典信息传递给game\_page和渲染器。

exit\_win(): 用户按下红x后的处理。使用messagebox窗体控件弹出询问窗口，是否确认退出。若确认退出，处于游戏界面则返回到menu\_page；处于其他界面，则关闭整个程序，需注意要关闭网络连接器connect。

7.1.2 class LoginPage

LoginPage类是客户端界面的登录页面，继承自tkinter.Frame类，并且继承组织窗体控件的功能。LoginPage的一个实例作为MainForm的重要组成部分。派生增加了属性有：指向主窗体的网络连接器指针connect；Entry对象entry\_id和entry\_password为用户名和密码的输入框（注意密码输入框要设定.show为’\*’字符）；Button对象button\_login和button\_regist为登录和注册按钮。

LoginPage的方法有：

login()：实现登录功能，该函数要与button\_login绑定，当button\_login被单击时则触发事件执行该函数。先对用户名和密码预检测，若用户名或密码为空，或者用户名ID不是合法数字则弹出messagebox消息提示错误。若合法，则向服务器发送登录指令，等待服务器返回。若服务器返回正确指令，则登录成功，调用主窗体的.toMenuPage()方法转至主菜单页面；否则弹出消息框报错。

regist()：实现注册功能，该函数要与button\_regist绑定，当button\_regist被单击时则触发事件执行该函数。首先弹出输入窗口，等待用户输入新账号和密码，用户确认后向服务器发送注册命令。若命令返回值无异常则弹出“完成注册”的消息框，否则弹出“注册错误”的消息框并显示详细错误信息。

7.1.3 class MenuPage

MenuPage类是客户端界面的主页面，继承自tkinter.Frame类，并且继承组织窗体控件的功能。MenuPage的一个实例作为MainForm的重要组成部分。派生增加的属性有：指向主窗体的网络连接器指针connect；显示等待动画的Canvas画布对象canvas（canvas初始化应设定为不可见状态）；Button对象button\_start为开始匹配按钮；以及相关成员属性，如icons\_load为动画每一帧组成的列表，\_icon\_index 为记录动画当前显示的帧下标int变量，waiting为记录是否处于匹配中状态的bool变量（初始化为False）。

MenuPage的方法有：

game\_wait()：实现“开始匹配”功能，该函数要与button\_start绑定，当button\_start被单击时则触发事件执行该函数。算法流程为判断当前是waiting是否为True，若为真（表明单击按键欲取消匹配），则设定waiting为False，canvas对象不可见，按键文字为“开始匹配”。否则（waiting为False），开始匹配，设定waiting为True，canvas对象可见，按键文字为“取消匹配”，并调用\_change\_icon()播放动画，向服务器发送开始匹配指令，并调用\_try\_start()尝试接收服务器的响应。

\_change\_icon()：实现播放等待动画功能。若处于等待状态（waiting为True），则\_icon\_index = (\_icon\_index + 1) % 12，把下一帧设为当前帧，并把新当前帧的画面绘制于canvas中。0.1s后产生事件再次运行该函数，以实现动画效果。

\_try\_start()：尝试接收服务器响应。若处于等待匹配状态，则尝试获取socket返回的数据。若socket的确接收到数据，且表明已加入游戏，则向服务器发送请求初始地图，并调用\_trygetinitmap()接收初始地图数据。若未接收到数据，则过5ms后生成事件再次运行该函数，尝试获取服务器响应。

\_trygetinitmap()：接收到初始地图。函数内不断循环，直到connect返回地图初始信息，取消等待状态（设定waiting为False，等待动画canvas不可见，按钮文字），并调用主窗体的toGamePage(dict)跳转至游戏页面，dict参数为刚接收到的初始地图数据。

7.1.4 class GamePage

GamePage类是客户端界面的游戏页面，继承自tkinter.Frame类，并且继承组织窗体控件的功能。GamePage的一个实例作为MainForm的重要组成部分。派生增加的属性有：指向主窗体的网络连接器指针connect；Canvas对象canvas\_main为显示游戏画面的画布；mapdisplay为游戏画面渲染器；key\_down为一个字典，格式为{‘Up’: 0|1, ‘Down’: 0|1, ‘Left’: 0|1, ‘Right’: 0|1, ‘fire’: 0|1}，记录哪些按键在这一帧内被按下；Label对象label\_hp, label\_bullet, label\_kill, label\_rank分别记录并显示玩家生命值、子弹数、杀敌数、排名；game\_end为记录游戏是否结束的bool变量。

GamePage的方法有：

game\_start(dict, int)：新游戏在客户端的初始化操作。参数dict为初始地图数据的字典，int为玩家ID。初始化流程为：设定game\_end为False，生成新的mapdisplay对象并初始化；绑定按键事件，即当用户按下按键时触发事件执行key\_handler(event)函数。并且触发事件执行\_game()函数和\_key\_trans()函数。

key\_handler(event)：按键句柄绑定函数，负责记录用户按键。参数event为用户按键事件，记录用户按键内容。当用户按下W或者上键，则key\_down的’Up’字段设为1；当用户按下W或者↑键，则key\_down的’Up’字段设为1；当用户按下S或者↓键，则key\_down的’Down’字段设为1；当用户按下A或者←键，则key\_down的’Left’字段设为1；当用户按下D或者→键，则key\_down的’Up’字段设为1；当用户按下空格或者回车键，则key\_down的’fire’字段设为1。

\_game()：游戏主循环，负责接收服务器的UDP实时地图信息，完成绘制。算法流程为：读取connector的所有实时地图变化信息，调用mapdisplay.changedict(dict)将变化信息传递给mapdisplay对象，让mapdisplay对象绘制当前帧的游戏画面和小地图画面，将绘制的游戏画面显示于canvas\_main中，并调用\_readplayer\_info(list[list])更新4个Label的显示值。最后根据服务器的实时地图信息判断游戏是否结束，若结束则调用ending()函数处理游戏后事宜，否则生成事件再次调用\_game()函数，循环显示当前游戏画面。

\_key\_trans()：向服务器发送用户操作的函数，循环不断发送最新操作。该函数需要精确控制单次循环时间为30ms，实现方式为每30ms生成一个事件执行该函数。若游戏未结束（game\_end为False），则检查key\_down字典是否有字段值不为0；若存在不为0的字段，则通过connect向服务器发送用户操作，并把key\_down的所有字段恢复为0。

\_readplayer\_info(list[list])：负责从所有玩家信息中找到自己的信息，并更新Label标签。参数为2维列表，即4.2.3中’tanks’字段的内容；第一层每个元素为一个玩家的信息，第二层记录该玩家的ID，血量，生命值，杀敌数等信息。算法流程为：遍历这个list的每个子列表，若这个子列表的list[0]（玩家ID）是自己的ID，则读取这个子列表的血量，生命值，杀敌数，更新标签，并退出遍历；玩家排名标签更改为当前剩余玩家数。

ending()：游戏结束后的善后工作。流程为：设定game\_end为true，取消按键绑定事件，显示消息框messagebox显示玩家杀敌数、排名，最后调用主窗体的toMenuPage()回到主菜单页面。

7.2 网络传输模块（client\_connector模块）

网络传输模块类图见“client\_connector类图.vsdx”。本模块依赖于Python自带库的网络套接字socket，多线程管理库threading，时间与休眠库time。依赖的其他库为该工程的打包解包库DataTransPack。

7.2.1 class Connector

Connector是客户端与服务器网络连接的类，该类的一个实例将成为主窗口的组成成员。该类的主成员诞生、初始化于主窗体所在线程，而UDP接收、发送消息，TCP接收、发送消息的循环分别在另外4个线程中。

Connector的成员有：

* \_serverIP：字符串记录的服务器IP地址。
* \_udpPort和\_tcpPort：int，服务器的UDP和TCP端口号。
* tcpsocket和udpsocket：socket套接字对象，分别用于TCP协议发送和接收、UDP协议发送和接收。
* inque\_udp, outque\_udp, inque\_tcp, outque\_tcp：4个队列Queue对象，分别记录UDP客户端向服务器发送消息，服务器返回数据；TCP客户端发送消息，服务器返回数据。
* udpsendth, udplistenth, tcpsendth, tcplistenth：4个线程Thread，分别执行函数\_thread\_udpsend(), \_thread\_udplisten(), \_thread\_tcpsend(), \_thread\_tcplisten()，以循环的方式不断监听服务器返回数据、客户端待发送的数据。
* tcp\_start：记录TCP相关的2个线程tcpsendth, tcplistenth是否打开，初始化时应为False。
* end\_flag：bool变量，记录是否已关闭connector，若为True，则将等待所有线程结束后关闭连接。初始化应为False。

Connector的方法有：

* udpStart()：打开运行和UDP相关的2个线程udpsendth, udplistenth。
* tcp\_link()：打开TCP的2个线程tcpsendth, tcplistenth，并建立与服务器的TCP连接。记得标记tcp\_start为True。
* tcp\_unlink()：断开与服务器的TCP的连接，但不结束线程。
* \_thread\_udpsend()：向服务器发送消息的循环，在udpsendth线程中执行。判断当end\_flag为False时循环，检查inque\_udp是否为空，若不为空则取出该队列的数据，使用udpsocket用UDP方式把数据发送至服务器。记得每次循环后sleep 5ms。
* \_thread\_udplisten()：监听从服务器发来的UDP协议数据，在udplistenth线程中执行。当end\_flag为False时循环，若从udpsocket读取到数据，则把数据放入outque\_udp中。每次循环后sleep 5ms。
* \_thread\_tcpsend()：向服务器发送消息的循环，在tcpsendth线程中执行。判断当end\_flag为False时循环，检查inque\_tcp是否为空，若不为空则取出该队列的数据，使用tcpsocket用TCP方式把数据发送至服务器。每次循环后sleep 5ms。
* \_thread\_tcplisten()：监听从服务器发来的TCP协议数据，在tcplistenth线程中执行。当end\_flag为False时循环，若从tcpsocket读取到数据，则把数据放入outque\_tcp中。每次循环后sleep 5ms。
* send\_data\_tcp(dict)：以TCP方式发送数据给服务器。参数dict为客户端需发送的字典。算法为：按照DataTransPack的客户端数据打包函数pack\_client\_data(dict)将字典打包为二进制，而后将二进制结果放入inque\_tcp队列中。
* send\_data\_udp(dict)：以UDP方式发送数据给服务器。参数dict为客户端需发送的字典。算法为：按照DataTransPack的客户端数据打包函数pack\_client\_data(dict)将字典打包为二进制，而后将二进制结果放入inque\_udp队列中。
* get\_tcp\_data()：获得从服务器返回的TCP协议数据。算法为：检查outque\_tcp是否为空，若不为空则取出二进制数据，按照DataTransPack的解包方式还原为Python字典，并返回该字典；若队列为空，则返回None。
* get\_udp\_data()：获得从服务器返回的UDP协议数据。算法为：检查outque\_udp是否为空，若不为空则取出二进制数据，按照DataTransPack的解包方式还原为Python字典，并返回该字典；若队列为空，则返回None。
* end()：抛弃所有数据，关闭该connect。实现过程为设置end\_flag为True。