**网络吃鸡版坦克大战：**

**客户端单元测试、集成测试计划文档**

负责人：胡俊杰

**1 测试简介**

该测试文档针对的是“**网络吃鸡版坦克大战**”项目的客户端各模块，包括client\_win，client\_connector，ClientDisplay三个模块的单元测试和集成测试。为了更高效的测试客户端，该测试还使用了核心game\_core作为辅助产生数据的模块。

该测试目的为检查客户端上述3个模块的功能性需求是否满足需求分析的要求，性能、稳定性等非功能需求是否达到用户的要求。

测试的策略为增量式测试，自下而上的测试方式。先测试模块内每个类、每个函数的功能是否达到要求，而后模块间组装集成，分步多次集成并不断测试集成后是否仍然满足需求。

测试的平台为笔记本开发环境。CPU为I5-8300H，4核心8线程，内存8GB。操作系统为Windows 10，测试平台IDE为PyCharm 64bit。

**2 单元测试**

2.1 client\_win模块测试

2.1.1 LoginPage测试：

GUI测试：

* 创建一个LoginPage对象并运行，观察GUI界面是否满足要求。预期结果：能看到用户名输入框、密码输入框、登录按钮、注册按钮。
* 向用户名输入框输入内容，观察界面效果，并设置观察点检查实际数据内容。预期结果：输入用户名时，用户名输入栏实时显示输入结果，但其他输入框不变。断点观察得到实际内容与显示相同。
* 向密码输入框输入内容，观察界面效果，并设置观察点检查实际数据内容。预期结果：输入密码时，密码框显示等长的字符串，但显示均为字符’\*’。设置断点观察实际数据，结果应与输入相同。

功能测试：

* 分别在用户名密码均不为空、用户名输入框为空、用户名不空但密码为空时点击“登录按钮”。预期结果：用户名密码均不空时完成初步验证，用户名为空时弹出消息框提示“错误：用户名不能为空”，密码为空时弹出消息框提示“错误：密码不能为空”。
* 点击“注册”按钮，观察程序表现。并试图输入新用户名和密码，只输入新用户名、只输入密码、全为空时确认注册。预期结果：点击“注册”后弹出新窗口，新窗口内有新用户名、密码输入框，以及确认注册的窗口。若用户名密码不全，则点击确认注册时弹出消息框提示错误。
* 点击红x按钮，观察界面效果，并在IDE中检查进程、内存。预期结果：程序关闭，GUI界面消失，进程结束，内存被释放。

2.1.2 MenuPage测试：

GUI测试：

* 创建MenuPage对象并运行，观察界面是否满足要求。预期结果：界面中央显示一个按钮，文字为“开始匹配”。页面下方显示设定好的用户名。

功能测试：

* 在非等待模式时点击“开始匹配”按钮，观察现象。并设置断点观察对象内部数据。预期结果：点击“开始匹配”后，画布显示并以动画方式不断播放加载动画，按钮文字变为“取消匹配”。内部数据结构显示waiting标记为True，\_icon\_index的值随着时间推移不断变化。
* 在等待匹配时再次点击按钮（此时按钮文字为“取消匹配”），观察现象，并设置断点观察内部数据。预期结果：动画消失，画布隐藏，waiting标记为False，\_icon\_index不再随时间变化。
* 用户按下红x，观察程序响应。预期结果：弹出消息框询问是否退出程序，若确定侧程序退出，界面消失，进程结束；若取消退出则恢复原状。

2.1.3 GamePage测试：

GUI测试：

* 创建GamePage对象并运行，观察界面是否满足要求。预期结果：界面中央为画布，下方有Label标签栏显示用户的生命值、弹药数、杀敌数、排名。

功能测试：

* 模拟玩家按下按键，让程序在\_key\_trans函数中输出操作，以测试按键事件绑定是否成功。按键测试用例和预期结果如表1.

表1 按键测试用例

|  |  |
| --- | --- |
| 按键测试用例 | 预期结果 |
| W键 | Key\_down的Up字段变为1，输出Up |
| S键 | Key\_down的Down字段变为1，输出Down |
| A键 | Key\_down的Left字段变为，输出Left |
| D键 | Key\_down的Right字段变为1，输出Right |
| 空格 | Key\_down的fire字段变为1，输出Fire |
| F键 | Key\_down字典无变化，无输出 |
| Ctrl键 | Key\_down字典无变化，无输出 |
| Esc键 | Key\_down字典无变化，无输出 |
| 不按下按键 | Key\_down字典无变化，无输出 |

* 用户点击红x试图退出游戏，观察程序响应。预期结果：弹出消息框询问是否确认退出，若点击确认退出则弹出“游戏退出”消息框，显示用户的杀敌数、排名，执行到断点ending()，否则恢复原状继续游戏。
* 模拟服务器传入实时变化的地图参数，传入的字典内容为：

{'info':[0,0,2,1,2,1],

'tanks':[[1,50,30,0,18,5.32,4.52],[2,80,40,1,108,9.32,12.52]],

'bulls':[[1,22,0,7.7,5.5]],

'obs':[[2,3], [8,8]],

'props':[[1,1,1,7,9],[2,1,1,5,6]],

'safe':[2,2,12,12,3,3,9,9]}

并设置玩家ID为2，观察界面底部玩家信息。预期结果：底部显示玩家生命值为80，子弹数40，杀敌数1，排名2。

性能测试：

* 长按某个按键，让程序不断输出，观察输出按键的流畅程度，并计时相邻按键之间的间隔时长。预期结果：输出按键较为流畅，时长间隔较为均匀，且在10ms内。

2.2 client\_connector模块测试

功能测试：

* 执行udpStart()，tcp\_link()，在IDE中检查线程状态，函数堆栈中寻找执行的函数。预期结果：4个线程udpsendth, udplistenth, tcpsendth, tcplistenth被执行，函数堆栈中找到4个线程执行的主函数。
* 调用send\_data\_tcp(dict), send\_data\_udp(dict)，设置断点观察队列outque\_tcp, outque\_udp。发送的内容为：

{‘id’: 4, ‘begin\_game’: 1}

预期结果：2个队列首先观察到不为空，数据被放入队列；而后队列又变为空，队列内容被函数\_thread\_tcpsend(), \_thread\_udpsend()取出。\_thread\_tcpsend(), \_thread\_udpsend()函数内成功读取json格式的打包结果。

* 调用get\_tcp\_data(), get\_udp\_data()，设置断点观察队列内容和返回值。在直接调用函数、模拟服务器发送消息后调用进行对比。预期结果：直接调用函数时，队列inque\_tcp, inque\_udp为空，函数返回值为None；模拟服务器发送消息后队列不为空，调用函数后返回服务器发送的数据，队列又变为空。
* 调用end()函数，在IDE中观察线程状态、内存状态。预期结果：4个线程被关闭，内存占用下降，得到释放。

性能测试：

* 向模拟服务器发送消息，接收消息，使用计时工具time记录本地消息从一个节点发送到另一个节点接收的用时。数据包大小有较小的json（1KB以下），较大的二进制流（30KB左右）。预期结果：对于较小的数据表，传输很快，在10ms以内完成。较大的二进制流也能在1s内传输完毕。

2.3 ClientDisplay单元测试

功能测试：

* 创建一个ClientDisplay对象，并向ClientDisplay初始化设定玩家ID为4，初始地图信息如下：

{ 'info': [0, -1, 1, 1, 2, 0, 2],

'tanks': [[2, 2, 3, 4, 40, 0, 1], [4, 2, 3, 4, 40, 6, 7]],

'bulls': [[1, 1, 1, 7, 7]],

'obs': [[1, 1], [4, 4]],

'props': [[1,0,1,7,9]],

'safe': [1, 1, 80, 80, 10, 10, 60, 60],

'glass': [[0, 0], [5, 5]] }

设置断点观察ClientDisplay.mapdict。预期结果：mapdict观察到传入的参数。

* 在测试1（上面的初始化信息）基础上，调用Draw()绘制主画面，调用SmallMap()绘制小地图画面，观察画面结果。预期结果：主画面的中间偏左上处有一个蓝色的坦克，朝向右上方；蓝色坦克的左上方有一个红色的坦克，朝向与蓝色坦克相同；蓝色坦克下方有一颗子弹，绘制爆炸效果；蓝色坦克右下角有一个血包道具；蓝色坦克左上角有2个障碍物，2个草丛，坐标对应。小地图中央有一个绿色正方形，外围包围着黄色正方形；黄色正方形左上角有一个红圆点。
* 调用函数changedict(dict)，传入参数内容如下：

{ 'info': [0, -1, 1, 1, 2, 0, 2],

'tanks': [[2, 2, 3, 4, 300, 0, 0], [4, 2, 3, 4, 90, 8, 8]],

'bulls': [[1, 2, 0, 3, 3]],

'obs': [[4, 4]],

'props': [[1,0,0,7,9], [2,1,1,5,6]],

'safe': [5, 5, 70, 70, 10, 10, 60, 60] }

设置断点观察mapdict。预期结果：mapdict的内容如下：

{ 'info': [0, -1, 1, 1, 2, 0, 2],

'tanks': [[2, 2, 3, 4, 300, 0, 0], [4, 2, 3, 4, 90, 8, 8]],

'bulls': [[1, 2, 1, 2, 2]],

'obs': [[1, 1]], // [4, 4]障碍物被击打

'props': [[2,1,1,5,6]], // 1个道具消失，1个道具产生

'safe': [5, 5, 70, 70, 10, 10, 60, 60] }

* 在调用changedict后，再次绘制主画面和小地图画面，观察画面结果。预期 结果：蓝色坦克向右下方移动，朝向转至右方；红色坦克消失（因为躲进草丛隐藏了）；蓝色坦克和原来红色坦克之间有一发子弹；离蓝色坦克较近的障碍物消失（被击毁）；蓝色坦克右下方的血包道具消失，而在左上方产生一个弹药箱道具。小地图中，绿色正方形未改变，黄色正方形缩小，向绿色正方形靠近，红色圆点向右下方移动。

性能测试：

* 不断循环调用Draw()和SmallMap()，并使用计时工具time记录每次调用2个函数的用时之和。预期结果：函数运行用时在30ms以下，且用时较为稳定。设置用例，当mapdict内容较多时，也尽可能使运行用时大于30ms的次数较少。

**3 集成测试**

3.1 客户端页面MainForm各个页面的集成

3.1.1 MainForm中含有LoginPage和MenuPage

* 正确输入账号密码，并使用模拟服务器发送登录成功的消息，观察界面变化。预期结果：客户端主界面从登录页面转为主菜单页面。

3.1.2 MainForm中聚合了LoginPage、MenuPage和GamePage

* 在MenuPage中点击“开始匹配按钮”，并使用模拟服务器发送成功匹配消息，观察界面变化。预期效果：客户端跳转至游戏页面GamePage。
* 在GamePage中点击红x，并确认退出，观察客户端界面。预期效果：弹出消息框显示用户游戏排名、杀敌数，确认后回到主菜单页面MenuPage。
* 在GamePage中，使用模拟服务器发送游戏结束消息，观察客户端界面。预期效果：弹出消息框显示用户游戏排名、杀敌数，确认后回到主菜单页面MenuPage。

3.2 MainForm集成ClientDisplay

* 在GamePage中模拟服务器输出地图信息，观察GamePage页面效果。预期：GamePage的画布上显示实时的游戏画面，画布左上角显示实时小地图，窗口下方Label标签处实时显示玩家当前状态（生命值、弹药数、杀敌数等）

3.3 客户端最终集成（加入client\_connector模块）

功能测试：

* 在LoginPage登录界面输入账号、密码，在connector设置断点观察队列值，connector连接到模拟服务器。测试用例：输入账号541，密码adc。预期效果：connector的outque\_tcp队列中出现json格式对象{‘id’: 541, ‘password’: ‘adc’}，而后队列中该元素被移除，发送至模拟服务器，模拟服务器成功接收到该json对象。
* 模拟服务器发送登录成功的信息：{‘id’: 541}，在connector设置断点，并观测客户端响应。预期效果：connector的tcpsocket接收到数据，并放入inque\_tcp队列中；而后队列内容被取出，GUI界面从登录界面转至主菜单界面。
* 在MenuPage中点击开始匹配按钮，在connector设置断点，输出模拟服务器接收到的数据。预期结果：connector的outque\_tcp队列中出现json格式对象{‘id’: 541, ‘begin\_game’: 1}，而后队列中该元素被移除，发送至模拟服务器，模拟服务器成功接收到该json对象。
* 模拟服务器返回成功匹配进入游戏的命令，在connector设置断点，观察客户端响应。预期结果：connector的tcpsocket接收到数据{‘id’: 541}，并放入inque\_tcp队列中；而后队列内容被取出，inque\_tcp增加初始地图的数据信息；最后初始地图的数据被取出，客户端界面跳转至GamePage游戏界面。
* 在GamePage中按下按键，按顺序分别按下W, A, S, D, 空格，在connector设置断点，并让模拟服务器输出接收到的内容。预期效果：队列outque\_udp中先后增加以下json对象：

{ ‘id’: 541, ‘Up’: 1, ‘Down’: 0, ‘Left’: 0, ‘Right’: 0, ‘fire’: 0 }

{ ‘id’: 541, ‘Up’: 0, ‘Down’: 0, ‘Left’: 1, ‘Right’: 0, ‘fire’: 0 }

{ ‘id’: 541, ‘Up’:0, ‘Down’: 1, ‘Left’: 0, ‘Right’: 0, ‘fire’: 0 }

{ ‘id’: 541, ‘Up’: 0, ‘Down’: 0, ‘Left’: 0, ‘Right’: 1, ‘fire’: 0 }

{ ‘id’: 541, ‘Up’: 0, ‘Down’: 0, ‘Left’: 0, ‘Right’: 0, ‘fire’: 1 }

而后队列的内容被逐个取出，发送至模拟服务器，模拟服务器成功接收玩家的按键数据。

* 模拟服务器不断发送实时地图信息，在connector设置断点观察队列，并观察客户端响应。预期结果：在connector的udpsocket接收到地图数据，并放入inque\_udp队列中；而后队列内容被取出，游戏界面画布、下方Label状态栏更新，且与传输的地图内容一致。

性能测试：

* 模拟服务器发送实时地图消息时，同时传输发送时间。客户端接收到数据、完成游戏画面渲染后记录完成时间。求从数据发送到完成渲染经过的时长。预期结果：时长一般情况能控制在50ms以内。