Chinese Chess

Validation Document

Author	
Group	**Team 5**
Date	**2022/6/4**

Catalog

1. Unit Test

This section provides information of unit tests we made for every function with statement coverage, branch coverage and condition coverage criteria. Testing cases with runnable test functions are provided in every test, you can find in corresponding files.

1.1 Controller

1.1.1 Restart Game

```
function restartGame(Obj)
    % Restart the game

    % Test Function:
    % tests/unitTestController/controllerBasic.m/testRestartGame

    % Reset status
    Obj.playerColorNow=1; % Statement Tcover1.1.1.1
    Obj.isChosen=0;
    Obj.hasMove=0;
    Obj.chosenChess=[0,0];
    Obj.lastMove=[0,0,0,0,0];
    Obj.lastEat=[0,0,0];

    % Reset chessboard
    Obj.chess=[8,6,4,3,1,2,5,7,9;
        0,0,0,0,0,0,0,0,0;
        0,10,0,0,0,0,0,11,0;
        12,0,14,0,16,0,13,0,15;
        0,0,0,0,0,0,0,0,0;
        0,0,0,0,0,0,0,0,0;
        32,0,30,0,31,0,29,0,28;
        0,27,0,0,0,0,0,26,0;
        0,0,0,0,0,0,0,0,0;
        25,23,21,19,17,18,20,22,24];

    % Reset UI
    Obj.chessBoard(1).restartGame();
    Obj.chessBoard(2).restartGame();
end
```

- Coverage Criteria: Statement coverage
- Test Function: tests/unitTestController/controllerBasic.m/testRestartGame
- Test Case

	Test Case T1.1.1.1
Input	-----
Coverage Item	Tcover1.1.1.1

State	c = controller;
	c.chess = [8,6,4,3,1,2,5,7,9;
	0,0,0,0,0,0,0,0,0;
	0,10,0,0,0,0,0,11,0;
	12,0,14,0,16,0,13,0,15;
	0,0,0,0,0,0,0,0,0;
	32,0,0,0,0,0,0,0,0;
	0,0,30,0,31,0,29,0,28;
	0,27,0,0,0,0,0,26,0;
	0,0,0,0,0,0,0,0,0;
	25,23,21,19,17,18,20,22,24];

Expected Output	The chess board is reset to default.
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.1.2 Change Player

```
function changePlayer(Obj)
    % Change round, change the playercolor and reset some ui

    % Test function:
    % tests/unitTestController/controllerBasic.m/testChangePlayer

    if Obj.playerColorNow==1 % Branch Tcover1.1.2.1
        Obj.playerColorNow=2;
        Obj.chessBoard(1).EndRound.Enable='off';
        Obj.chessBoard(1).Cancel.Enable='off';
        Obj.chessBoard(1).Revoke.Enable='on';
        Obj.chessBoard(2).Revoke.Enable='off';
        Obj.chessBoard(1).roundAnnounce.Text="对手回合";
        Obj.chessBoard(2).roundAnnounce.Text="你的回合";
    else % Branch Tcover1.1.2.2
        Obj.playerColorNow=1;
        Obj.chessBoard(2).EndRound.Enable='off';
        Obj.chessBoard(2).Cancel.Enable='off';
        Obj.chessBoard(2).Revoke.Enable='on';
        Obj.chessBoard(1).Revoke.Enable='off';
```

```
            Obj.chessBoard(2).roundAnnounce.Text="对手回合";
            Obj.chessBoard(1).roundAnnounce.Text="你的回合";
        end
        Obj.chessBoard(1).requestTimes=[0,0,0];
        Obj.chessBoard(1).EndGame.Enable='on';
        Obj.chessBoard(1).Restart.Enable='on';
        Obj.chessBoard(2).requestTimes=[0,0,0];
        Obj.chessBoard(2).EndGame.Enable='on';
        Obj.chessBoard(2).Restart.Enable='on';
        Obj.hasMove=0;
        Obj.isChosen=0;
        Obj.chosenChess=[0,0];
        Obj.RecordMove=[Obj.lastMove,Obj.lastEat(3)];
        % Obj.RecordMove
        % Obj.lastMove
        Obj.lastMove=[0,0,0,0,0];
        Obj.lastEat=[0,0,0];
        Obj.chessBoard(1).getChoice(4);
        Obj.chessBoard(2).getChoice(4);
    end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestController/controllerBasic.m/testChangePlayer
- Test Case

	Test Case T1.1.2.1	Test Case T1.1.2.2
Input	-----	-----
Coverage Item	Tcover1.1.2.1	Tcover1.1.2.2
State	c = controller; c.playerColorNow = 1;	c = controller; c.playerColorNow = 2;
Expected Output	Red Player shows “对手回合“; Black Player shows “你的回合”;	Red Player shows “你的回合“; Black Player shows “对手回合”;
Test Result	Passed	Passed

- Test Coverage: 2/2 = 100%

1.1.3 End Game

```
function endGame(Obj,loser)
    % The game end, announce result , when loser is 3 means the game
    % ends in a draw

    % Test Function:
    % tests/unitTestController/controllerBasic.m/testEndGame

    switch loser
```

```
            case 1 % Branch Tcover1.1.3.1
                Obj.chessBoard(1).announceResult(2);
                Obj.chessBoard(2).announceResult(1);
            case 2 % Branch Tcover1.1.3.2
                Obj.chessBoard(1).announceResult(1);
                Obj.chessBoard(2).announceResult(2);
            case 3 % Branch Tcover1.1.3.3
                Obj.chessBoard(1).announceResult(3);
                Obj.chessBoard(2).announceResult(3);
        end
    end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestController/controllerBasic.m/testEndGame
- Test Case

	Test Case T1.1.3.1	Test Case T1.1.3.2	Test Case T1.1.3.3
Input	1	2	3
Coverage Item	Tcover1.1.3.1	Tcover1.1.3.2	Tcover1.1.3.3
State	c = controller;	c = controller;	c = controller;
Expected Output	Red Player shows “失败”; Black Player shows “胜利”	Red Player shows “胜利”; Black Player shows “失败”	Red Player shows “平手”; Black Player shows “平手”
Test Result	Passed	Passed	Passed

- Test Coverage: 3/3 = 100%

1.1.4 Click At

```
function clickAt(Obj,x,y)
    % Player click at point [x,y],

    % Test function:
    % tests/unitTestController/controllerPlay.m/testClickAt

    if Obj.isChosen==0 % Branch Tcover1.1.4.1
        % Haven't chosen a chess,
        if Obj.chess(x,y)==0 % Branch Tcover1.1.4.2
            return
        end
        choseColor=1;
        if Obj.chess(x,y)>16 % Branch Tcover1.1.4.3
            choseColor=2;
        end
        if Obj.playerColorNow==choseColor % Branch Tcover1.1.4.4
            % Belong to this player,then chose it
```

```
                Obj.isChosen=1;
                Obj.chosenChess=[x,y];
                Obj.chessBoard(1).moveTarget(x,y);
                Obj.chessBoard(2).moveTarget(x,y);
                Obj.lastMove=[0,0,0,0,0];
                Obj.lastEat=[0,0,0];
            end
        else % Branch Tcover1.1.4.5
            if Obj.chosenChess(1)== x && Obj.chosenChess(2)==y
                % Branch Tcover1.1.4.6
                % Click at a chess twice, then unchose
                Obj.isChosen=0;
                Obj.chosenChess=[0,0];
                Obj.chessBoard(1).resetTarget;
                Obj.chessBoard(2).resetTarget;
                return
            end
            moveResult=Obj.checkMove(Obj.chosenChess(1), ...
                Obj.chosenChess(2),x,y);
            % Check if move valid
            if moveResult==1 % Branch Tcover1.1.4.7
                Obj.lastMove=[Obj.chosenChess(1), ...
                    Obj.chosenChess(2),x,y, ...
                    Obj.chess(Obj.chosenChess(1), ...
                    Obj.chosenChess(2))];
                Obj.lastEat=[x,y,Obj.chess(x,y)];
                if Obj.chess(x,y)~=0 % Branch Tcover1.1.4.8
                    Obj.chessBoard(1).removeChess(Obj.chess(x,y));
                    Obj.chessBoard(2).removeChess(Obj.chess(x,y));
                end
                Obj.chess(x,y)=Obj.lastMove(5);
                Obj.chess(Obj.chosenChess(1),Obj.chosenChess(2))=0;
                Obj.chessBoard(1).moveChess(Obj.lastMove(5),x,y);
                Obj.chessBoard(2).moveChess(Obj.lastMove(5),x,y);
                Obj.hasMove=1;
                Obj.isChosen=0;
                Obj.chosenChess=[0,0];
            end
        end
    end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestController/controllerPlay.m/testClickAt
- Test Case

	Test Case T1.1.4.1	Test Case T1.1.4.2	Test Case T1.1.4.3

Input	7, 2	7, 1	7.1
Coverage Item	Tcover1.1.4.1, Tcover1.1.4.2	Tcover1.1.4.1, Tcover1.1.4.3, Tcover1.1.4.4	Tcover1.1.4.5, Tcover1.1.4.6

State	c = controller;
	c.isChosen == 0;
	c.playerColorNow == 2;
		c = controller;
		c.isChosen == 0;
		c.playerColorNow == 2;
			c = controller;
			c.isChosen == 1;
			c.playerColorNow == 2;
			c.chosenChess == [7, 1];

Expected Output	Target point is not highlighted.	Target point is highlighted.	Target point highlight disappears.
Test Result	Passed	Passed	Passed
	Test Case T1.1.4.4		
Input	6, 1		
Coverage Item	Tcover1.1.4.5, Tcover1.1.4.7, Tcover1.1.4.8		

State	c = controller;
	c.isChosen == 1;
	c.playerColorNow == 2;
	c.chosenChess == [7, 1];		

Expected Output	The chess piece is successfully moved.		
Test Result	Passed		

- Test Coverage: 8/8 = 100%

1.1.5 Cancel Move

```
function cancelMove(Obj)
    % Cancel the move in this round

    % Test function:
    % tests/unitTestController/controllerPlay.m/testCancelMove

    Obj.hasMove=0;
    Tx=Obj.lastMove(1);
    Ty=Obj.lastMove(2);
    Fx=Obj.lastMove(3);
    Fy=Obj.lastMove(4);
    Num=Obj.lastMove(5);
    Obj.chess(Tx,Ty)=Num;
    Obj.chess(Fx,Fy)=0;
    Obj.chessBoard(1).moveChess(Num,Tx,Ty);
    Obj.chessBoard(2).moveChess(Num,Tx,Ty);
```

```
            if Obj.lastEat(3)~=0 % Branch Tcover 1.1.5.1
                Obj.chess(Fx,Fy)=Obj.lastEat(3);
                Obj.chessBoard(1).replaceChess(Obj.lastEat(3));
                Obj.chessBoard(2).replaceChess(Obj.lastEat(3));
            end
        end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestController/controllerPlay.m/testCancelMove
- Test Case

	Test Case T1.1.5.1
Input	-----
Coverage Item	Tcover1.1.5.1

State	c = controller;
	c.isChosen == 1;
	c.playerColorNow == 2;
	c.chosenChess == [7, 1];

Expected Output	The move is canceled, and the chess piece is reset.
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.1.6 Back to Last Round

```
        function backToLastRound(Obj)
            % Revoke, which will back to last round, need to reset
            % some chess.

            % Test function:
            % tests/unitTestController/controllerPlay.m/testBackToLastRound

            if Obj.hasMove % Branch Tcover1.1.6.1
                Obj.cancelMove();
                % If in this round the player has move, need to
                % cancel move first
            end
            Num=Obj.RecordMove(5);
            Tx=Obj.RecordMove(1);
            Ty=Obj.RecordMove(2);
            Fx=Obj.RecordMove(3);
            Fy=Obj.RecordMove(4);
            eat=Obj.RecordMove(6);
            Obj.chess(Tx,Ty)=Num;
            Obj.chess(Fx,Fy)=eat;
            Obj.chessBoard(1).moveChess(Num,Tx,Ty);
```

```
            Obj.chessBoard(2).moveChess(Num,Tx,Ty);
            if Obj.chess(Fx,Fy)~=0 % Branch Tcover1.1.6.2
                Obj.chessBoard(1).replaceChess(eat);
                Obj.chessBoard(2).replaceChess(eat);
            end
            Obj.changePlayer();
        end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestController/controllerPlay.m/testBackToLastRound
- Test Case

	Test Case T1.1.6.1
Input	-----
Coverage Item	Tcover1.1.6.1, Tcover1.1.6.2

State	c = controller;
	c.chess = [8,6,4,3,1,2,5,7,9;
	0,0,0,0,0,0,0,0,0;
	0,10,0,0,0,0,0,11,0;
	12,0,14,0,0,0,13,0,15;
	0,0,0,0,16,0,0,0,0;
	0,0,0,0,31,0,0,0,0;
	32,0,30,0,0,0,29,0,28;
	0,27,0,0,0,0,0,26,0;
	0,0,0,0,0,0,0,0,0;
	25,23,21,19,17,18,20,22,24]
	c.playerColorNow = 1;
	c.clickAt(5, 5);
	c.clickAt(6, 5);
	c.changePlayer();
	c.clickAt(7, 9);
	c.clickAt(6, 9);

Expected Output	The move is canceled, and the chess piece is reset.
Test Result	Passed

- Test Coverage: 2/2 = 100%

1.1.7 Check Win

```
        function checkWin(Obj)
            % Check if game end.

            % Test function:
            % tests/unitTestController/controllerPlay.m/testCheckWin
```

```
        if Obj.lastEat(3)==1 % Branch Tcover1.1.7.1
            Obj.endGame(1);
        end
        if Obj.lastEat(3)==17 % Branch Tcover1.1.7.2
            Obj.endGame(2);
        end
    end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestController/controllerPlay.m/testCheckWin
- Test Case

	Test Case T1.1.7.1	Test Case T1.1.7.2
Input	-----	-----
Coverage Item	Tcover1.1.7.1	Tcover1.1.7.2
State	c = controller; c.lastEat(3) == 1;	c = controller; c.lastEat(3) == 17;
Expected Output	Black Player wins.	Red Player wins.
Test Result	Passed	Passed

- Test Coverage: 2/2 = 100%

1.1.8 Check Move

```
function T=checkMove(Obj,Fx,Fy,Tx,Ty)
    % Check if move valid

    % Test function:
    % tests/unitTestController/controllerRRules.m/*
    % tests/unitTestController/controller.BRules.m/*
    % *We seperate test cases by the type of chess piece.

    T=0;
    switch Obj.chess(Fx,Fy)
        case 1 % Branch Tcover1.1.8.1
            % Red king
            % Test function:
            % tests/unitTestController/controllerRRules.m/testRKing
            if Ty>=4 && Ty<=6 && Tx<=3 && ...
                abs(Tx-Fx)+abs(Ty-Fy)==1 && ...
                (Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)>16)
                % Branch Tcover1.1.8.2
                % Can't eat the same color chess
                T=1;
            end
        case {2,3} % Branch Tcover1.1.8.3
```

```
        % Shi
        % Test function:
        % tests/unitTestController/controllerRRules.m/testRShi
        if Ty>=4 && Ty<=6 && Tx<=3 && ...
            (abs(Tx-Fx)==1)&&(abs(Ty-Fy)==1) && ...
            (Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)>16)
            % Branch Tcover1.1.8.4
            T=1;
        end
    case {4,5} % Branch Tcover1.1.8.5
        % Xiang
        % Test function:
        % tests/unitTestController/controllerRRules.m/testRXiang
        if Tx<6 && (abs(Tx-Fx)==2)&&(abs(Ty-Fy)==2) && ...
            Obj.chess((Tx+Fx)/2,(Ty+Fy)/2)==0 && ...
            (Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)>16)
            % Branch Tcover1.1.8.6
            T=1;
        end
    case {6,7} % Branch Tcover1.1.8.7
        % Ma
        % Test function:
        % tests/unitTestController/controllerRRules.m/testRMa
        if (abs(Tx-Fx)+abs(Ty-Fy)==3) ...
            &&abs(abs(Tx-Fx)-abs(Ty-Fy))==1
            if (abs(Tx-Fx)==2 && Obj.chess((Tx+Fx)/2,Fy)~=0) ...
               || (abs(Ty-Fy)==2 && Obj.chess(Fx,(Ty+Fy)/2)~=0)
                % Branch Tcover1.1.8.8
                return
            end
            if Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)>16
                % Branch Tcover1.1.8.9
                T=1;
            end
        end
    case {8,9} % Branch Tcover1.1.8.10
        % Che
        % Test function:
        % tests/unitTestController/controllerRRules.m/testRChe
        if (Tx-Fx)~=0&&(Ty-Fy)~=0 % Branch Tcover1.1.8.11
            return
        end
        if (Ty-Fy)==0 % Branch Tcover1.1.8.12
            for k=(min(Tx, Fx)+1):(max(Tx, Fx)-1)
```

```
                if Obj.chess(k,Fy)~=0 % Branch Tcover1.1.8.13
                    return
                end
            end
        else % Branch Tcover1.1.8.14
            for k=(min(Ty, Fy)+1):(max(Ty, Fy)-1)
                if Obj.chess(Fx,k)~=0 % Branch Tcover1.1.8.15
                    return
                end
            end
        end
        if Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)>16
            % Branch Tcover1.1.8.16
            T=1;
        end
    case {10,11} % Branch Tcover1.1.8.17
        % Pao
        % Test function:
        % tests/unitTestController/controllerRRules.m/testRPao
        if (Tx-Fx)~=0&&(Ty-Fy)~=0 % Branch Tcover1.1.8.18
            return
        end
        if Obj.chess(Tx,Ty)==0 % Branch Tcover1.1.8.19
            if (Ty-Fy)==0 % Branch Tcover1.1.8.20
                for k=(min(Tx, Fx)+1):(max(Tx, Fx)-1)
                    if Obj.chess(k,Fy)~=0
                        % Branch Tcover1.1.8.21
                        return
                    end
                end
            else % Branch Tcover1.1.8.22
                for k=(min(Ty, Fy)+1):(max(Ty, Fy)-1)
                    if Obj.chess(Fx,k)~=0
                        % Branch Tcover1.1.8.23
                        return
                    end
                end
            end
            T=1;
        else % Branch Tcover1.1.8.24
            counter=0;
            if (Ty-Fy)==0 % Branch Tcover1.1.8.25
                for k=(min(Tx, Fx)+1):(max(Tx, Fx)-1)
                    if Obj.chess(k,Fy)~=0
```

```
                    % Branch Tcover1.1.8.26
                    counter=counter+1;
                end
            end
        else % Branch Tcover1.1.8.27
            for k=(min(Ty, Fy)+1):(max(Ty, Fy)-1)
                if Obj.chess(Fx,k)~=0
                    % Branch Tcover1.1.8.28
                    counter=counter+1;
                end
            end
        end
        if counter~=1 % Branch Tcover1.1.8.29
            return
        end
        if Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)>16
            % Branch Tcover1.1.8.30
            T=1;
        end
    end
case {12,13,14,15,16} % Branch Tcover1.1.8.31
    % Zu
    % Test function:
    % tests/unitTestController/controllerRRules.m/testRZu
    if Tx<Fx || abs(Tx-Fx)+abs(Ty-Fy)>1 ...
        || (Fx<6 && Fx==Tx) % Branch Tcover1.1.8.32
        % Can't do that!!!
        return
    end
    if Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)>16
        % Branch Tcover1.1.8.33
        T=1;
    end
case 17 % Branch Tcover1.1.8.34
    % Black king
    % Test function:
    % tests/unitTestController/controllerBRules.m/testBKing
    if Ty>=4 && Ty<=6 && Tx>=8 && ...
       abs(Tx-Fx)+abs(Ty-Fy)==1 && ...
        (Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)<17)
        % Branch Tcover1.1.35
        T=1;
    end
case {18,19} % Branch Tcover1.1.8.36
```

```
        % Shi
        % Test function:
        % tests/unitTestController/controllerBRules.m/testBShi
        if Ty>=4 && Ty<=6 && Tx>=8 && ...
           (abs(Tx-Fx)==1)&&(abs(Ty-Fy)==1) && ...
           (Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)<17)
            % Branch Tcover1.1.8.37
            T=1;
        end
    case {20,21} % Branch Tcover1.1.8.38
        % Xiang
        % Test function:
        % tests/unitTestController/controllerBRules.m/testBXiang
        if Tx>5 && (abs(Tx-Fx)==2)&&(abs(Ty-Fy)==2) && ...
           Obj.chess((Tx+Fx)/2,(Ty+Fy)/2)==0 && ...
           (Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)<17)
            % Branch Tcover1.1.8.39
            T=1;
        end
    case {22,23} % Branch Tcover1.1.8.40
        % Ma
        % Test function:
        % tests/unitTestController/controllerBRules.m/testBMa
        if (abs(Tx-Fx)+abs(Ty-Fy)==3)&& ...
            abs(abs(Tx-Fx)-abs(Ty-Fy))==1
            if (abs(Tx-Fx)==2 && Obj.chess((Tx+Fx)/2,Fy)~=0)...
               || (abs(Ty-Fy)==2 && Obj.chess(Fx,(Ty+Fy)/2)~=0)
                % Branch Tcover1.1.8.41
                return
            end
            if Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)<17
                % Branch Tcover1.1.8.42
                T=1;
            end
        end
    case {24,25} % Branch Tcover1.1.8.43
        % Che
        % Test function:
        % tests/unitTestController/controllerBRules.m/testBChe
        if (Tx-Fx)~=0&&(Ty-Fy)~=0
            % Branch Tcover1.1.8.44
            return
        end
        if (Ty-Fy)==0
```

```
            % Branch Tcover1.1.8.45
            for k=(min(Tx, Fx)+1):(max(Tx, Fx)-1)
                if Obj.chess(k,Fy)~=0
                    % Branch Tcover1.1.8.46
                    return
                end
            end
        else % Branch Tcover1.1.8.47
            for k=(min(Ty, Fy)+1):(max(Ty, Fy)-1)
                if Obj.chess(Fx,k)~=0
                    % Branch Tcover1.1.8.48
                    return
                end
            end
        end
        if Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)<17
            % Branch Tcover1.1.8.49
            T=1;
        end
    case {26,27} % Branch Tcover1.1.8.50
        % Pao
        % Test function:
        % tests/unitTestController/controllerBRules.m/testBPao
        if (Tx-Fx)~=0&&(Ty-Fy)~=0
            % Branch Tcover1.1.8.51
            return
        end
        if Obj.chess(Tx,Ty)==0 % Branch Tcover1.1.8.52
            if (Ty-Fy)==0 % Branch Tcover1.1.8.53
                for k=(min(Tx, Fx)+1):(max(Tx, Fx)-1)
                    if Obj.chess(k,Fy)~=0
                        % Branch Tcover1.1.8.54
                        return
                    end
                end
            else % Branch Tcover1.1.8.55
                for k=(min(Ty, Fy)+1):(max(Ty, Fy)-1)
                    if Obj.chess(Fx,k)~=0
                        % Branch Tcover1.1.8.56
                        return
                    end
                end
            end
            T=1;
```

```
                else % Branch Tcover1.1.8.57
                    counter=0;
                    if (Ty-Fy)==0 % Branch Tcover1.1.8.58
                        for k=(min(Tx, Fx)+1):(max(Tx, Fx)-1)
                            if Obj.chess(k,Fy)~=0
                                % Branch Tcover1.1.8.59
                                counter=counter+1;
                            end
                        end
                    else % Branch Tcover1.1.8.60
                        for k=(min(Ty, Fy)+1):(max(Ty, Fy)-1)
                            if Obj.chess(Fx,k)~=0
                                % Branch Tcover1.1.8.61
                                counter=counter+1;
                            end
                        end
                    end
                    if counter~=1 % Branch Tcover1.1.8.62
                        return
                    end
                    if Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)<17
                        % Branch Tcover1.1.8.63
                        T=1;
                    end
                end
            case {28,29,30,31,32} % Branch Tcover1.1.8.64
                % Zu
                % Test function:
                % tests/unitTestController/controllerBRules.m/testBZu
                if Fx<Tx || abs(Tx-Fx)+abs(Ty-Fy)>1 ...
                   || (Fx>5 && Fx==Tx) % Branch Tcover1.1.8.65
                    %can't do that!!!
                    return
                end
                if Obj.chess(Tx,Ty)==0 || Obj.chess(Tx,Ty)<17
                    % Branch Tcover1.1.8.66
                    T=1;
                end
        end
    end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestController/controllerRRules.m/*,
  tests/unitTestController/controllerBRules.m/*
- Test Case

	Test Case T1.1.8.1	Test Case T1.1.8.2	Test Case T1.1.8.3
Input	1, 5, 2, 5	1, 5, 2, 2	1, 4, 2, 5
Coverage Item	Tcover1.1.8.1, Tcover1.1.8.2	Tcover1.1.8.1, Tcover1.1.8.2	Tcover1.1.8.3, Tcover1.1.8.4
State	c = controller; c.chess = default;	c = controller; c.chess = default;	c = controller; c.chess = default;
Expected Output	T == 1	T == 0	T == 1
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.4	Test Case T1.1.8.5	Test Case T1.1.8.6
Input	1, 4, 2, 4	1, 3, 3, 1	1, 3, 3, 3
Coverage Item	Tcover1.1.8.3, Tcover1.1.8.4	Tcover1.1.8.5, Tcover1.1.8.6	Tcover1.1.8.5, Tcover1.1.8.6
State	c = controller; c.chess = default;	c = controller; c.chess = default;	c = controller; c.chess = default;
Expected Output	T == 0	T == 1	T == 0
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.7	Test Case T1.1.8.8	Test Case T1.1.8.9
Input	1, 3, 3, 1	1, 2, 3, 3	1, 2, 3, 3
Coverage Item	Tcover1.1.8.5, Tcover1.1.8.6	Tcover1.1.8.7, Tcover1.1.8.9	Tcover1.1.8.7, Tcover1.1.8.8
State	c = controller; c.chess = [8,6,4,3,1,2,5,7,9; 0,10,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24]; Xiang is blocked and cannot move.	c = controller; c.chess = default;	c = controller; c.chess = [8,6,4,3,1,2,5,7,9; 0,10,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24]; Ma is blocked and cannot move.
Expected Output	T == 0	T == 1	T == 0
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.10	Test Case T1.1.8.11	Test Case T1.1.8.12
Input	1, 1, 2, 2	1, 1, 1, 3	1, 1, 5, 1
Coverage Item	Tcover1.1.8.10, Tcover1.1.8.11	Tcover1.1.8.10, Tcover1.1.8.12,	Tcover1.1.8.10, Tcover1.1.8.14,

		Tcover1.1.8.13	Tcover1.1.8.15
State	c = controller; c.chess = default;	c = controller; c.chess = default;	c = controller; c.chess = default;
Expected Output	T == 0	T == 0	T == 0
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.13	Test Case T1.1.8.14	Test Case T1.1.8.15
Input	1, 1, 2, 1	3, 2, 2, 1	3, 2, 9, 2
Coverage Item	Tcover1.1.8.10, Tcover1.1.8.16	Tcover1.1.8.17, Tcover1.1.8.18	Tcover1.1.8.17, Tcover1.1.8.19, Tcover1.1.8.20, Tcover1.1.8.21
State	c = controller; c.chess = default;	c = controller; c.chess = default;	c = controller; c.chess = default;
Expected Output	T == 0	T == 0	T == 0
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.16	Test Case T1.1.8.17	Test Case T1.1.8.18
Input	3, 2, 3, 9	3, 2, 3, 3	7, 2, 7, 7
Coverage Item	Tcover1.1.8.17, Tcover1.1.8.19, Tcover1.1.8.22, Tcover1.1.8.23	Tcover1.1.8.17, Tcover1.1.8.19	Tcover1.1.8.17 Tcover1.1.8.24, Tcover1.1.8.25, Tcover1.1.8.26, Tcover1.1.8.29
State	c = controller; c.chess = default;	c = controller; c.chess = default;	c = controller; c.chess = [... 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 32,10,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24];
Expected Output	T == 0	T == 1	T == 0
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.19	Test Case T1.1.8.20	Test Case T1.1.8.21
Input	6, 3, 10, 3	4, 1, 4, 2	4, 1, 5, 1
Coverage Item	Tcover1.1.8.17, Tcover1.1.8.24,	Tcover1.1.8.31, Tcover1.1.8.32	Tcover1.1.8.31, Tcover1.1.8.33

	Tcover1.1.8.27, Tcover1.1.8.28, Tcover1.1.8.30		

State	c = controller;
	c.chess = [...
	8,6,4,3,1,2,5,7,9;
	0,0,0,0,0,0,0,0,0;
	0,0,0,0,0,0,0,11,0;
	12,0,14,0,16,0,13,0,15;
	0,0,0,0,0,0,0,0,0;
	0,0,10,0,0,0,0,0,0;
	32,0,30,0,31,0,29,0,28;
	0,27,0,0,0,0,0,26,0;
	0,0,0,0,0,0,0,0,0;
	25,23,21,19,17,18,20,22,24];
		c = controller;
		c.chess = default;
			c = controller;
			c.chess = default;
			Zu is not allowed to turn left or right before crossed the river.

Expected Output	T == 1	T == 1	T == 0
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.22	Test Case T1.1.8.23	Test Case T1.1.8.24
Input	6, 5, 6, 4	6, 5, 5, 5	10, 5, 9, 5
Coverage Item	Tcover1.1.8.31, Tcover1.1.8.33	Tcover1.1.8.31, Tcover1.1.8.32	Tcover1.1.8.34, Tcover1.1.8.35
State	c = controller; c.chess = [... 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,10,0,0,0,0,0,11,0; 12,0,14,0,0,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,16,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24];	c = controller; c.chess = [... 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,10,0,0,0,0,0,11,0; 12,0,14,0,0,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,16,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24]; Zu is not allowed to move backward.	c = controller; c.chess = default;
Expected Output	T == 1	T == 0	T == 1
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.25	Test Case T1.1.8.26	Test Case T1.1.8.27
Input	10, 5, 9, 6	10, 6, 9, 5	10, 6, 9, 6
Coverage Item	Tcover1.1.8.34, Tcover1.1.8.35	Tcover1.1.8.36, Tcover1.1.8.37	Tcover1.1.8.36, Tcover1.1.8.37
State	c = controller;	c = controller;	c = controller;

	c.chess = default;	c.chess = default;	c.chess = default;
Expected Output	T == 0	T == 1	T == 0
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.28	Test Case T1.1.8.29	Test Case T1.1.8.30
Input	10, 7, 8, 9	10, 7, 8, 7	10, 7, 8, 9
Coverage Item	Tcover1.1.8.38, Tcover1.1.8.39	Tcover1.1.8.38, Tcover1.1.8.39	Tcover1.1.8.38, Tcover1.1.8.39
State	c = controller; c.chess = default;	c = controller; c.chess = default;	c = controller; c.chess = [… 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,10,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,26,0; 25,23,21,19,17,18,20,22,24] Xiang is blocked and cannot move.
Expected Output	T == 1	T == 0	T == 0
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.31	Test Case T1.1.8.32	Test Case T1.1.8.33
Input	10, 8, 8, 7	10, 8, 8, 7	10, 9, 9, 8
Coverage Item	Tcover1.1.8.40, Tcover1.1.8.42	Tcover1.1.8.40, Tcover1.1.8.41	Tcover1.1.8.43, Tcover1.1.8.44
State	c = controller; c.chess = default;	c = controller; c.chess = [… 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,10,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,26,0; 25,23,21,19,17,18,20,22,24]; Ma is blocked and cannot move.	c = controller; c.chess = default;
Expected	T == 1	T == 0	T == 0

Output			
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.34	Test Case T1.1.8.35	Test Case T1.1.8.36
Input	10, 9, 10, 7	10, 9, 6, 9	10, 9, 9, 9
Coverage Item	Tcover1.1.8.43, Tcover1.1.8.45, Tcover1.1.8.46	Tcover1.1.8.43, Tcover1.1.8.47, Tcover1.1.8.48	Tcover1.1.8.43, Tcover1.1.8.49
State	c = controller; c.chess = default;	c = controller; c.chess = default;	c = controller; c.chess = default;
Expected Output	T == 0	T == 0	T == 1
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.37	Test Case T1.1.8.38	Test Case T1.1.8.39
Input	8, 8, 9, 9	8, 8, 2, 8	8, 8, 8, 1
Coverage Item	Tcover1.1.8.50, Tcover1.1.8.51	Tcover1.1.8.50, Tcover1.1.8.52, Tcover1.1.8.53, Tcover1.1.8.54	Tcover1.1.8.50, Tcover1.1.8.52, Tcover1.1.8.55, Tcover1.1.8.56
State	c = controller; c.chess = default;	c = controller; c.chess = default;	c = controller; c.chess = default;
Expected Output	T == 0	T == 0	T == 0
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.40	Test Case T1.1.8.41	Test Case T1.1.8.42
Input	8, 8, 8, 7	4, 8, 4, 3	5, 7, 5, 1
Coverage Item	Tcover1.1.8.50, Tcover1.1.8.52,	Tcover1.1.8.50, Tcover1.1.8.57, Tcover1.1.8.58, Tcover1.1.8.59, Tcover1.1.8.62	Tcover1.1.8.50, Tcover1.1.8.57, Tcover1.1.8.60, Tcover1.1.8.61, Tcover1.1.8.63
State	c = controller; c.chess = default;	c = controller; c.chess = [... 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,10,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,26,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24];	c = controller; c.chess = [... 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,0,0,26,0,0; 0,0,10,0,0,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24];
Expected	T == 1	T == 0	T == 1

Output			
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.43	Test Case T1.1.8.44	Test Case T1.1.8.45
Input	7, 9, 7, 8	7, 9, 6, 9	5, 5, 5, 6
Coverage Item	Tcover1.1.8.64, Tcover1.1.8.65	Tcover1.1.8.64, Tcover1.1.8.66	Tcover1.1.8.64, Tcover1.1.8.66

State	c = controller;
	c.chess = default;
	Zu is not allowed to turn left or right before crossing the river.
		c = controller;
		c.chess = default
			c = controller;
			c.chess = […
			8,6,4,3,1,2,5,7,9;
			0,0,0,0,0,0,0,0,0;
			0,10,0,0,0,0,0,11,0;
			12,0,14,0,16,0,13,0,15;
			0,0,0,0,31,0,0,0,0;
			0,0,0,0,0,0,0,0,0;
			32,0,30,0,0,0,29,0,28;
			0,27,0,0,0,0,0,26,0;
			0,0,0,0,0,0,0,0,0;
			25,23,21,19,17,18,20,22,24];

Expected Output	T == 0	T == 1	T == 1
Test Result	Passed	Passed	Passed
	Test Case T1.1.8.46		
Input	5, 5, 6, 5		
Coverage Item	Tcover1.1.8.64, Tcover1.1.8.65		
State	c = controller; c.chess =[… 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,10,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,31,0,0,0,0; 0,0,0,0,0,0,0,0,0; 32,0,30,0,0,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24]; Zu is not allowed to move backward.		
Expected Output	T == 0		
Test Result	Passed		

- Test Coverage: 66/66 = 100%

1.2 Chessboard UI

1.2.1 Position to Number

```
function [nx,ny]=pos2No(app,x,y)
    % Change position to num

    % Test function:
    % tests/unitTestChessboard/chessboardBasic.m/testPos2No

    minDis=1000;
    minNum=0;
    x=x-25;
    y=y-25;
    for i = 1:10
        if minDis>abs(app.chessPosX(i)-x) % Branch T1.2.1.1
            minDis=abs(app.chessPosX(i)-x);
            minNum=i;
        end
    end
    nx=minNum;
    minDis=1000;
    minNum=0;
    for i = 1:9
        if minDis>abs(app.chessPosY(i)-y) % Branch T1.2.1.2
            minDis=abs(app.chessPosY(i)-y);
            minNum=i;
        end
    end
    ny=minNum;

    if app.playerColor==1 % Branch T1.2.1.3
        nx=11-nx;
        ny=10-ny;
    end
end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestChessboard/chessboardBasic/testPos2No
- Test Case

	Test Case T1.2.1.1	Test Case T1.2.1.2
Input	140, 275	140, 275
Coverage Item	Tcover1.2.1.1,	Tcover1.2.1.1,

	Tcover1.2.1.2 Tcover1.2.1.3	Tcover1.2.1.2
State	c = controller; c.playerColorNow = 1;	c = controller; c.playerColorNow = 2;
Expected Output	3, 8	8, 2
Test Result	Passed	Passed

- Test Coverage: 2/2 = 100%

1.2.2 Number to Position

```
function [x,y]=no2Pos(app,nx,ny,Type)
            % Change num to position

            % Test function:
            % tests/unitTestChessboard/chessboardBasic.m/testNo2Pos

%             nx
%             ny
            if app.playerColor==1 % Branch Tcover1.2.2.1
                nx=11-nx;
                ny=10-ny;
            end
            if Type==1 % Branch Tcover1.2.2.2
                x=app.chessPosX(nx);
                y=app.chessPosY(ny);
            else % Branch Tcover1.2.2.3
                x=app.chessPosX(nx);
                y=app.chessPosY(ny);
                x=x-10;
                y=y-10;
            end
        end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestChessboard/chessboardBasic/testNo2Pos
- Test Case

	Test Case T1.2.2.1	Test Case T1.2.2.2
Input	3, 8, 1	3, 8, 2
Coverage Item	Tcover1.2.2.1, Tcover1.2.2.2	Tcover1.2.2.1, Tcover1.2.2.3
State	c = controller; c.playerColorNow = 1;	c = controller; c.playerColorNow = 1;
Expected Output	119, 241	119, 231
Test Result	Passed	Passed

- Test Coverage: 2/2 = 100%

1.2.3 Get Choice

```
function getChoice(app,type)
    % Show the text when the player make request

    % Test function:
    % tests/unitTestChessboard/chessboardBasic.m/testGetChoice

    switch type
        case 1 % Branch Tcover1.2.3.1
            app.Label.Text="对手请求重新开始";
        case 2 % Branch Tcover1.2.3.2
            app.Label.Text="对手请求和棋";
        case 3 % Branch Tcover1.2.3.3
            app.Label.Text="对手请求悔棋";
        case 4 % Branch Tcover1.2.3.4
            app.choiceState=0;
            app.Label.Visible='off';
            app.Accept.Visible='off';
            app.Refuse.Visible='off';
            return
    end
    app.choiceState=type;
    app.Label.Visible='on';
    app.Accept.Visible='on';
    app.Refuse.Visible='on';
end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestChessboard/chessboardBasic/testGetChoice
- Test Case

	Test Case T1.2.3.1	Test Case T1.2.3.2	Test Case T1.2.3.3
Input	1	2	3
Coverage Item	Tcover1.2.3.1	Tcover1.2.3.2	Tcover1.2.3.3
State	c = controller; c.playerColorNow = 1;	c = controller; c.playerColorNow = 1;	c = controller; c.playerColorNow = 1;
Expected Output	Red Player shows “对手请求重新开始”.	Red Player shows “对手请求和棋”.	Red Player shows “对手请求悔棋”.
Test Result	Passed	Passed	Passed
	Test Case T1.2.3.4		
Input	4		
Coverage Item	Tcover1.2.3.4		

State	c = controller; c.playerColorNow = 1;		
Expected Output	c.chessBoard(1).choiceStat e == 0;		
Test Result	Passed		

- Test Coverage: 4/4 = 100%

1.2.4 Announce Result

```
function announceResult(app,winner)
        % Announce who win and makesure player can't do some
        % thing unpredictable.

        % Test functions:
        % tests/unitTestChessboard/chessboardBasic.m/testAnnounceResult

        switch winner
            case 1 % Branch Tcover1.2.4.1
                app.Result.Text='胜利';
            case 2 % Branch Tcover1.2.4.2
                app.Result.Text='失败';
            case 3 % Branch Tcover1.2.4.3
                app.Result.Text='平手';
        end
        app.Label.Visible="off";
        app.Accept.Visible="off";
        app.Refuse.Visible="off";
        app.roundAnnounce.Text="游戏结束";
        app.endGame=1;
        app.Result.Visible='on';
        app.EndRound.Enable='off';
        app.EndGame.Enable='off';
        app.Cancel.Enable='off';
        app.Revoke.Enable='off';
        app.Surrender.Enable='off';
end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestChessboard/chessboardBasic/testAnnounceResult
- Test Case

	Test Case T1.2.4.1	Test Case T1.2.4.2	Test Case T1.2.4.3
Input	1	2	3
Coverage Item	Tcover1.2.4.1	Tcover1.2.4.2	Tcover1.2.4.3

State	c = controller; c.playerColorNow = 1;	c = controller; c.playerColorNow = 1;	c = controller; c.playerColorNow = 1;
Expected Output	Red Player shows “胜利”.	Red Player shows “失败”.	Red Player shows “平手”.
Test Result	Passed	Passed	Passed

- Test Coverage: 3/3 = 100%

1.2.5 Restart Game

```
function restartGame(app)
    % Restart, reset all the data

    % Test functions:
    % tests/unitTestChessboard/chessboardBasic.m/testRestartGame

    for i=1:10
        for j=1:9
            if app.controller.chess(i,j) ==0
                % Branch Tcover1.2.5.1
                continue
            end
            if app.controller.chess(i,j)<17
                % Branch Tcover1.2.5.2
                [PosX,PosY]=app.no2Pos(i,j,1);
                app.chessPos(2)=PosX;
                app.chessPos(1)=PosY;
                app.chessHandle(1,app.controller.chess(i, ...
                    j)).Position=app.chessPos;
                app.chessHandle(1,app.controller.chess(i, ...
                    j)).Visible='on';
            end
            if app.controller.chess(i,j)>16
                % Branch Tcover1.2.5.3
                [PosX,PosY]=app.no2Pos(i,j,1);
                app.chessPos(2)=PosX;
                app.chessPos(1)=PosY;
                app.chessHandle(2,app.controller.chess(i, ...
                    j)-16).Position=app.chessPos;
                app.chessHandle(2,app.controller.chess(i, ...
                    j)-16).Visible='on';
            end
        end
    end
    app.endGame=0;
```

```
            app.requestTimes=[0,0,0];
            app.resetTarget();
            app.Result.Visible='off';
            app.EndRound.Enable='off';
            app.EndGame.Enable='on';
            app.Cancel.Enable='off';
            app.Surrender.Enable='on';
            app.Label.Visible="off";
            app.Accept.Visible='off';
            app.Refuse.Visible='off';
            app.Revoke.Enable='off';
            switch app.playerColor
                case 1 % Branch Tcover1.2.5.4
                    app.roundAnnounce.Text="你的回合";
                case 2 % Branch Tcover1.2.5.5
                    app.roundAnnounce.Text="对手回合";
            end
            app.choiceState=0;
        end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestChessboard/chessboardBasic/testRestartGame
- Test Case

	Test Case T1.2.5.1
Input	-----
Coverage Item	Tcover1.2.5.1, Tcover1.2.5.2, Tcover1.2.5.3, Tcover1.2.5.4, Tcover1.2.5.5
State	c = controller;
Expected Output	Red Player shows “你的回合”; Black Player shows “对手回合”;
Test Result	Passed

- Test Coverage: 5/5 = 100%

1.2.6 Reset Target

```
        function resetTarget(app)
            % Reset the chosen target ico.

            % Test functions:
            % tests/unitTestChessboard/chessboardPlay.m/testRemove

            app.targetPos(1)=-80; % Statement Tcover1.2.6.1
```

```
        app.targetPos(2)=-80;
        app.Target.Position=app.targetPos;
    end
```

- Coverage Criteria: Statement coverage
- Test Function: tests/unitTestChessboard/chessboardPlay/testResetTarget
- Test Case

	Test Case T1.2.6.1
Input	-----
Coverage Item	Tcover1.2.6.1
State	c = controller;
Expected Output	Target (highlight) position is updated.
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.2.7 Move Target

```
    function moveTarget(app,x,y)
        % Move the target indicator

        % Test functions:
        % tests/unitTestChessboard/chessboardPlay.m/testMoveTarget

        [PosX,PosY]=app.no2Pos(x,y,2); % Statement Tcover1.2.7.1
        app.targetPos(2)=PosX;
        app.targetPos(1)=PosY;
        app.Target.Position=app.targetPos;
        app.Target.Visible='on';
    end
```

- Coverage Criteria: Statement coverage
- Test Function: tests/unitTestChessboard/chessboardPlay/testMoveTarget
- Test Case

	Test Case T1.2.7.1
Input	1, 1
Coverage Item	Tcover1.2.7.1
State	c = controller;
Expected Output	Target (highlight) position is set to the right position.
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.2.8 Remove Chess

```
    function removeChess(app,Num)
```

```
        % Eat a chess, then remove it

        % Test functions:
        % tests/unitTestChessboard/chessboardPlay.m/testRemoveChess

        chessColor=1;
        if Num>16 % Branch Tcover1.2.8.1
            Num=Num-16;
            chessColor=2;
        end
        app.chessHandle(chessColor,Num).Visible="off";
    end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestChessboard/chessboardPlay/testRemoveChess
- Test Case

	Test Case T1.2.8.1	Test Case T1.2.8.2
Input	24	8
Coverage Item	Tcover1.2.8.1	Tcover1.2.8.1
State	c = controller;	c = controller;
Expected Output	The target chess piece is removed.	The target chess piece is removed.
Test Result	Passed	Passed

- Test Coverage: 1/1 = 100%

1.2.9 Replace Chess

```
    function replaceChess(app,Num)
        % Cancel move, then replace it

        % Test functions:
        % tests/unitTestChessboard/chessboardPlay.m/testReplaceChess

        chessColor=1;
        if Num>16 % Tcover1.2.9.1
            Num=Num-16;
            chessColor=2;
        end
        app.chessHandle(chessColor,Num).Visible="on";
    end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestChessboard/chessboardPlay/testReplaceChess
- Test Case

	Test Case T1.2.9.1	Test Case T1.2.9.2

Input	24	8
Coverage Item	Tcover1.2.9.1	Tcover1.2.9.1
State	c = controller;	c = controller;
Expected Output	The target chess piece is removed and replaced.	The target chess piece is removed and replaced.
Test Result	Passed	Passed

- Test Coverage: 2/2 = 100%

1.2.10 Move Chess

```
function moveChess(app,Num,Tx,Ty)
    % Move the image of chess

    % Test functions:
    % tests/unitTestChessboard/chessboardPlay.m/testMoveChess

    chessColor=1;
    if Num>16 % Branch Tcover1.2.10.1
        Num=Num-16;
        chessColor=2;
    end
    [PosX,PosY]=no2Pos(app,Tx,Ty,1);
    app.chessPos(2)=PosX;
    app.chessPos(1)=PosY;
    app.resetTarget();
    app.chessHandle(chessColor,Num).Position=app.chessPos;
    app.chessPos=[0,0,50,50];
    if app.playerColor~=app.controller.playerColorNow
        % Branch Tcover1.2.10.2
        return
    end
    app.Cancel.Enable="on";
    app.EndRound.Enable="on";
end
```

- Coverage Criteria: Branch coverage
- Test Function: tests/unitTestChessboard/chessboardPlay/testMoveChess
- Test Case

	Test Case T1.2.10.1	Test Case T1.2.10.2
Input	8, 2, 1	31, 5, 5
Coverage Item	Tcover1.2.10.1	Tcover1.2.10.1, Tcover1.2.10.2
State	c = controller;	c = controller; c.chess = [...

		8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,10,0,0,0,0,0,11,0; 12,0,14,0,0,0,13,0,15; 0,0,0,0,16,0,0,0,0; 0,0,0,0,31,0,0,0,0; 32,0,30,0,0,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24];
Expected Output	The target chess piece is moved to the expected position.	The target chess piece is moved to the expected position.
Test Result	Passed	Passed

- Test Coverage: 2/2 = 100%

2. Integration Test

This section provides information of integration tests we made for the Chinese Chess. Since the system only has two components, its integration test is not complex. Testing cases with runnable test functions are provided in every test, you can find in corresponding files.

2.1 Controller + Chessboard

- Test function: tests/integrationTest/chessGameInt.m/testIntAll
- Test Case

	Test Case T2.1
Operation	Move Pao from (3, 2) to (3, 5) from Red Player. Click end round button from Red Player. Move Zu from (7, 5) to (6, 5) from Black Player. Click restart button from Black Player. Click refuse button from Red Player. Click revoke button from Red Player. Click refuse button from Black Player. Click end round button from Black Player. Move Zu from (4, 5) to (5, 5) from Red Player. Click end round button from Red Player. Move Che from (10, 1) to (9, 1) from the Black Player. Click cancel button from Black Player. Move Zu from (6, 5) to (5, 5) from Black Player. Click end round button from Black Player. Move Pao from (3, 5) to (10, 5) from Red Player.

	Click end round button from Red Player. Click restart button from Black Player. Click accept button from Red Player. Move Pao from (3, 2) to (3, 5) from Red Player. Press surrender button form Red Player.
Coverage Item	Tcover1.1.1.1, Tcover1.1.2.1, Tcover1.1.2.2, Tcover1.1.3.1, Tcover1.1.4.1, Tcover1.1.4.3, Tcover1.1.4.4, Tcover1.1.5.1, Tcover1.1.4.5, Tcover1.1.4.7, Tcover1.1.4.8, Tcover1.1.7.2, Tcover1.1.8.17, Tcover1.1.8.19, Tcover1.1.8.22, Tcover1.1.8.24, Tcover1.1.8.27, Tcover1.1.8.30, Tcover1.1.8.31, Tcover1.1.8.33, Tcover1.1.8.64, Tcover1.1.8.66, Tcover1.2.1.1, Tcover1.2.1.2, Tcover1.2.1.3, Tcover1.2.2.1, Tcover1.2.2.2, Tcover1.2.2.3, Tcover1.2.3.2, Tcover1.2.5.1, Tcover1.2.5.2, Tcover1.2.5.3, Tcover1.2.5.4, Tcover1.2.5.5, Tcover1.2.7.1, Tcover1.2.8.1, Tcover1.2.9.1, Tcover1.2.10.1,

Expected Output	In the first round, Red Player wins by strategy. In the second round, Black Player wins by surrender.
Test Result	Passed

- Test Coverage: 38/38 = 100%

3. Functionality Test

This section provides information of functionality tests we made for the Chinese Chess with commonly seen use cases. Testing cases with runnable test functions are provided in every test, you can find in corresponding files.

3.1 Use Case “Restart the game”

3.1.1 Test “Ask for restart then be rejected”

- Test Function: tests/functionalityTest/chessGameBasic.m/testRestart
- Test Case

	Test Case T3.1.1
Operation	Press restart button from the Red Player, and press reject button from the Black Player.
State	Chessboard is not the default.
Expected Behavior	A dialog appears and after being rejected by the enemy, the game is not reset and remains unchanged.
Test Result	Passed

3.1.2 Test “Ask for restart then be accepted”

- Test Function: tests/functionalityTest/chessGameBasic.m/testRestart
- Test Case

	Test Case T3.1.2
Operation	Press restart button from the Red Player, and press accept button from the Black Player.
State	Chessboard is not the default.
Expected Behavior	Chessboard is set to default and all the labels and buttons are reset to the status that they are in when a game starts.
Test Result	Passed

3.1.3 Test “Ask for restart for three times and be unable to ask for restart for more times”

- Test Function: tests/functionalityTest/chessGameBasic.m/testRestart
- Test Case

	Test Case T3.1.3
Operation	Press restart button from the Red Player, and press refuse button from the Black Player. Continue this for three times
State	Chessboard is not the default.
Expected Behavior	Restart button from the Red Player is disabled.
Test Result	Passed

3.2 Use Case “Ask for peace”

3.2.1 Test “Ask for peace then be rejected”

- Test Function: tests/functionalityTest/chessGameBasic.m/testPeace
- Test Case

	Test Case T3.2.1
Operation	Press peace button from the Red Player, and press reject button from the Black Player.
State	Chessboard is the default.
Expected Behavior	A dialog appears and after being rejected by the enemy, the dialog disappears, and nothing happens.
Test Result	Passed

3.2.2 Test “Ask for peace then be accepted”

- Test Function: tests/functionalityTest/chessGameBasic.m/testPeace
- Test Case

	Test Case T3.2.2
Operation	Press peace button from the Red Player, and press accept button from the Black Player.
State	Chessboard is the default.
Expected Behavior	A dialog appears and after being accepted by the enemy, the game ends with result showing “平手” on both sides.
Test Result	Passed

3.2.3 Test “Ask for peace for three times and be unable to ask for peace for more times”

- Test Function: tests/functionalityTest/chessGameBasic.m/testPeace
- Test Case

	Test Case T3.2.3
Operation	Press peace button from the Red Player, and press refuse button from the Black Player. Continue this for three times
State	Chessboard is the default.
Expected Behavior	Peace button from the Red Player is disabled.
Test Result	Passed

3.3 Use Case “Surrender”

3.3.1 Test “Surrender”

- Test Function: tests/functionalityTest/chessGameBasic.m/testSurrender
- Test Case

	Test Case T3.3.1
Operation	Press surrender button from the Red Player.
State	Chessboard is the default.
Expected Behavior	The game ends with result showing “失败” on the Red Player and “胜利” on the Black Player.
Test Result	Passed

3.4 Use Case “Cancel last move”

3.4.1 Test “Cancel last move”

- Test Function: tests/functionalityTest/chessGameBasic.m/testCancel
- Test Case

	Test Case T3.4.1
Operation	Move Zu from (4, 1) to (5, 1) and press cancel button from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	Last move is canceled, and the chessboard is reset.
Test Result	Passed

3.5 Use Case “Ask for revoke”

3.5.1 Test “Ask for revoke and be rejected”

- Test Function: tests/functionalityTest/chessGameBasic.m/testRevoke
- Test Case

	Test Case T3.5.1
Operation	Press revoke button from the Red Player, and press reject button from the Black Player.
State	Red Player has moved Zu from (4, 1) to (5, 1) in last round. In this round Black Player has moved Zu from (7, 1) to (6, 1).
Expected Behavior	A dialog appears and after being rejected by the enemy, the dialog disappears, and nothing happens.
Test Result	Passed

3.5.2 Test “Ask for revoke and be accepted”

- Test Function: tests/functionalityTest/chessGameBasic.m/testRevoke
- Test Case

	Test Case T3.5.2
Operation	Press revoke button from the Red Player, and press accept button from the Black Player.
State	Red Player has moved Zu from (4, 1) to (5, 1) in last round. In this round Black Player has moved Zu from (7, 1) to (6, 1).
Expected Behavior	A dialog appears and after being rejected by the enemy, the dialog disappears, and the chessboard is reset to last round.
Test Result	Passed

3.5.3 Test “Ask for revoke for three times and be unable to ask for peace for more times.”

- Test Function: tests/functionalityTest/chessGameBasic.m/testRevoke
- Test Case

	Test Case T3.5.3
Operation	Press revoke button from the Red Player, and press reject button from the Black Player. Continue this for three times
State	Red Player has moved Zu from (4, 1) to (5, 1) in last round. In this round Black Player has moved Zu from (7, 1) to (6, 1).

Expected Behavior	Revoke button from the Red Player is disabled.
Test Result	Passed

3.6 Use Case “Choose a chess piece”

3.6.1 Test “Choose an empty space”

- Test Function: tests/functionalityTest/chessGamePlay.m/testChooseChess
- Test Case

	Test Case T3.6.1
Operation	Click an empty space on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	Nothing happens.
Test Result	Passed

3.6.2 Test “Choose a chess piece from own side”

- Test Function: tests/functionalityTest/chessGamePlay.m/testChooseChess
- Test Case

	Test Case T3.6.2
Operation	Click a chess piece from own side on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The chess piece is highlighted.
Test Result	Passed

3.6.3 Test “Un-choose a chess piece of own side”

- Test Function: tests/functionalityTest/chessGamePlay.m/testChooseChess
- Test Case

	Test Case T3.6.3
Operation	Click the highlighted chess piece from own side on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn. A chess piece of own side is already chosen and highlighted.
Expected Behavior	The chess piece is un-highlighted.
Test Result	Passed

3.6.4 Test “Choose a chess piece when it is the enemy’s turn”

- Test Function: tests/functionalityTest/chessGamePlay.m/testChooseChess
- Test Case

	Test Case T3.6.4
Operation	Click a chess piece on the chessboard from the Black Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	Nothing happens
Test Result	Passed

3.7 Use Case “Move a chess piece”

This section provides testing results for moving the chess abide by and against the Chinese Chess rules.

3.7.1 Case “Move chess piece Wang”

3.7.1.1 Test “Move Wang abiding by the rules”

- Test Function: tests/functionalityTest/chessGameRules.m/testKing
- Test Case

	Test Case T3.7.1.1
Operation	Move Wang from (1, 5) to (2, 5) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is allowed and happens.
Test Result	Passed

3.7.1.2 Test “Move Wang for more than one step”

- Test Function: tests/functionalityTest/chessGameRules.m/testKing
- Test Case

	Test Case T3.7.1.2
Operation	Move Wang from (1, 5) to (3, 5) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is not allowed since Wang can only move one step at a time and noting happens.
Test Result	Passed

3.7.1.3 Test “Move Wang diagonally”

- Test Function: tests/functionalityTest/chessGameRules.m/testKing
- Test Case

	Test Case T3.7.1.3
Operation	Move Wang from (1, 5) to (2, 4) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is not allowed since Wang can only move along the grid and noting happens.
Test Result	Passed

3.7.1.4 Test “Move Wang to other piece from own side

- Test Function: tests/functionalityTest/chessGameRules.m/testKing
- Test Case

	Test Case T3.7.1.4
Operation	Move Wang from (1, 5) to (1, 4) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is not allowed since two pieces from the same side cannot stay in the same position and noting happens.
Test Result	Passed

3.7.1.5 Test “Move Wang out of the tent

- Test Function: tests/functionalityTest/chessGameRules.m/testKing
- Test Case

	Test Case T3.7.1.5
Operation	Move Wang from (2,4) to (2, 3) on the chessboard from the Red Player.
State	Chessboard is [… 8,6,4,3,0,2,5,7,9; 0,0,0,1,0,0,0,0,0; 0,10,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24]. It is the Red Player’s turn.
Expected Behavior	The move is not allowed since Wang cannot move out of the tent and noting

	happens.
Test Result	Passed

3.7.2 Case “Move chess piece Shi”

3.7.2.1 Test “Move Shi abiding by the rules”

- Test Function: tests/functionalityTest/chessGameRules.m/testShi
- Test Case

	Test Case T3.7.2.1
Operation	Move Shi from (1, 4) to (2, 5) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is allowed and happens.
Test Result	Passed

3.7.2.2 Test “Move Shi not diagonally”

- Test Function: tests/functionalityTest/chessGameRules.m/testShi
- Test Case

	Test Case T3.7.2.2
Operation	Move Shi from (1, 4) to (2, 4) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is not allowed since Shi can only move diagonally and nothing happens.
Test Result	Passed

3.7.2.3 Test “Move Shi for more than one step”

- Test Function: tests/functionalityTest/chessGameRules.m/testShi
- Test Case

	Test Case T3.7.2.3
Operation	Move Shi from (1, 4) to (3, 6) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is not allowed since Shi can only move one step at a time and nothing happens.
Test Result	Passed

3.7.2.4 Test “Move Shi out of the tent”

- Test Function: tests/functionalityTest/chessGameRules.m/testShi
- Test Case

	Test Case T3.7.2.4
Operation	Move Shi from (1, 4) to (2, 3) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is not allowed since Shi cannot move out of the tent and nothing happens.
Test Result	Passed

3.7.3 Case “Move chess piece Xiang”

3.7.3.1 Test “Move Xiang abiding by the rules”

- Test Function: tests/functionalityTest/chessGameRules.m/testXiang
- Test Case

	Test Case T3.7.3.1
Operation	Move Xiang from (1, 3) to (3, 1) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is allowed and happens.
Test Result	Passed

3.7.3.2 Test “Move Xiang not by ‘Tian’”

- Test Function: tests/functionalityTest/chessGameRules.m/testXiang
- Test Case

	Test Case T3.7.3.2
Operation	Move Xiang from (1, 3) to (3, 3) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is not allowed since Xiang can only move by “Tian” and nothing happens.
Test Result	Passed

3.7.3.3 Test “Move Xiang with barricades”

- Test Function: tests/functionalityTest/chessGameRules.m/testXiang
- Test Case

	Test Case T3.7.3.2

Operation	Move Xiang from (1, 3) to (3, 1) on the chessboard from the Red Player.
State	Chessboard is [… 8,6,4,3,1,2,5,7,9; 0,10,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24]. It is the Red Player's turn.
Expected Behavior	The move is not allowed since Xiang cannot move when there is a barricade in the center of the "Tian", and nothing happens.
Test Result	Passed

3.7.4 Case "Move chess piece Ma"

3.7.4.1 Test "Move Ma abiding by the rules"

- Test Function: tests/functionalityTest/chessGameRules.m/testMa
- Test Case

	Test Case T3.7.4.1
Operation	Move Ma from (1, 2) to (3, 3) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player's turn.
Expected Behavior	The move is allowed and happens.
Test Result	Passed

3.7.4.2 Test "Move Ma not by 'Ri'"

- Test Function: tests/functionalityTest/chessGameRules.m/testMa
- Test Case

	Test Case T3.7.4.2
Operation	Move Ma from (1, 2) to (2, 3) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player's turn.
Expected Behavior	The move is not allowed since Ma can only move by "Ri" and nothing happens.
Test Result	Passed

3.7.4.3 Test “Move Ma with barricades”

- Test Function: tests/functionalityTest/chessGameRules.m/testMa
- Test Case

	Test Case T3.7.4.3
Operation	Move Ma from (1, 2) to (3, 3) on the chessboard from the Red Player.
State	Chessboard is [… 8,6,4,3,1,2,5,7,9; 0,10,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24]. It is the Red Player’s turn.
Expected Behavior	The move is not allowed since Ma cannot move when there is a barricade in the “Ri” which is next to Ma, and nothing happens.
Test Result	Passed

3.7.5 Case “Move chess piece Che”

3.7.5.1 Test “Move Che abiding by the rules.”

- Test Function: tests/functionalityTest/chessGameRules.m/testChe
- Test Case

	Test Case T3.7.5.1
Operation	Move Che from (1, 1) to (3, 1) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is allowed and happens.
Test Result	Passed

3.7.5.2 Test “Move Che diagonally”

- Test Function: tests/functionalityTest/chessGameRules.m/testChe
- Test Case

	Test Case T3.7.5.2
Operation	Move Che from (1, 1) to (2, 2) on the chessboard from the Red Player.

State	Chessboard is the default. It is the Red Player's turn.
Expected Behavior	The move is not allowed since Che can only move diagonally and nothing happens.
Test Result	Passed

3.7.5.3 Test "Move Che with barricades"

- Test Function: tests/functionalityTest/chessGameRules.m/testChe
- Test Case

	Test Case T3.7.5.3
Operation	Move Che from (1, 1) to (5, 1) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player's turn.
Expected Behavior	The move is not allowed since Che cannot move when there is a barricade on the route, and nothing happens.
Test Result	Passed

3.7.6 Case "Move chess piece Pao"

3.7.6.1 Test "Move Pao abiding by the rules."

- Test Function: tests/functionalityTest/chessGameRules.m/testPao
- Test Case

	Test Case T3.7.6.1
Operation	Move Pao from (3, 2) to (5, 2) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player's turn.
Expected Behavior	The move is allowed and happens.
Test Result	Passed

3.7.6.2 Test "Move Pao diagonally"

- Test Function: tests/functionalityTest/chessGameRules.m/testPao
- Test Case

	Test Case T3.7.6.2
Operation	Move Pao from (3, 2) to (2, 1) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player's turn.
Expected Behavior	The move is not allowed since Pao can only move along the grid and nothing happens.
Test Result	Passed

3.7.6.3 Test “Move Pao with barricades”

- Test Function: tests/functionalityTest/chessGameRules.m/testPao
- Test Case

	Test Case T3.7.6.3
Operation	Move Pao from (3, 2) to (9, 2) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player’s turn.
Expected Behavior	The move is not allowed since Pao cannot move when there is a barricade on the route, and nothing happens.
Test Result	Passed

3.7.6.4 Test “Move Pao as a hit abiding by the rules”

- Test Function: tests/functionalityTest/chessGameRules.m/testPao
- Test Case

	Test Case T3.7.6.4
Operation	Move Pao from (6, 3) to (10, 3) on the chessboard from the Red Player.
State	Chessboard is [… 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,10,0,0,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24]. It is the Red Player’s turn.
Expected Behavior	The move is allowed and happens.
Test Result	Passed

3.7.6.5 Test “Move Pao as a hit with more than one barricade”

- Test Function: tests/functionalityTest/chessGameRules.m/testPao
- Test Case

	Test Case T3.7.6.5
Operation	Move Pao from (7, 2) to (7, 7) on the chessboard from the Red Player.
State	Chessboard is [… 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0;

	0,0,0,0,0,0,0,11,0; 12,0,14,0,16,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0,0; 32,10,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24]. It is the Red Player's turn.
Expected Behavior	The move is not allowed since Pao cannot hit with more than one barricade on the route and nothing happens.
Test Result	Passed

3.7.6.6 Test "Move Pao as a hit with no barricade"

- Test Function: tests/functionalityTest/chessGameRules.m/testPao
- Test Case

	Test Case T3.7.6.6
Operation	Move Pao from (3, 2) to (8, 2) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player's turn.
Expected Behavior	The move is not allowed since Pao cannot hit when there is no barricade on the route, and nothing happens.
Test Result	Passed

3.7.7 Case "Move chess piece Zu"

3.7.7.1 Test "Move Zu abiding by the rules"

- Test Function: tests/functionalityTest/chessGameRules.m/testZu
- Test Case

	Test Case T3.7.7.1
Operation	Move Pao from (4, 1) to (5, 1) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player's turn.
Expected Behavior	The move is allowed and happens.
Test Result	Passed

3.7.7.2 Test "Move Zu for more than one step"

- Test Function: tests/functionalityTest/chessGameRules.m/testZu
- Test Case

	Test Case T3.7.7.2
Operation	Move Pao from (4, 1) to (6, 1) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player's turn.
Expected Behavior	The move is not allowed since Zu can only move one step at a time and nothing happens.
Test Result	Passed

3.7.7.3 Test "Move Zu to turn left or right before crossing the river"

- Test Function: tests/functionalityTest/chessGameRules.m/testZu
- Test Case

	Test Case T3.7.7.3
Operation	Move Pao from (4, 1) to (4, 2) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player's turn.
Expected Behavior	The move is not allowed since Zu cannot turn left or right before crossing the river, and nothing happens.
Test Result	Passed

3.7.7.4 Test "Move Zu diagonally"

- Test Function: tests/functionalityTest/chessGameRules.m/testZu
- Test Case

	Test Case T3.7.7.4
Operation	Move Pao from (4, 1) to (5, 2) on the chessboard from the Red Player.
State	Chessboard is the default. It is the Red Player's turn.
Expected Behavior	The move is not allowed since Zu cannot move diagonally and nothing happens.
Test Result	Passed

3.7.7.5 Test "Move Zu to turn left or right after crossing the river."

- Test Function: tests/functionalityTest/chessGameRules.m/testZu
- Test Case

	Test Case T3.7.7.5
Operation	Move Pao from (6, 5) to (6, 4) on the chessboard from the Red Player.
State	Chessboard is [... 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,10,0,0,0,0,0,11,0; 12,0,14,0,0,0,13,0,15; 0,0,0,0,0,0,0,0,0;

	0,0,0,0,16,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24]. It is the Red Player's turn.
Expected Behavior	The move is allowed and happens.
Test Result	Passed

3.7.7.6 Test "Move Zu backward."

- Test Function: tests/functionalityTest/chessGameRules.m/testZu
- Test Case

	Test Case T3.7.7.6
Operation	Move Pao from (6, 5) to (5, 5) on the chessboard from the Red Player.
State	Chessboard is [... 8,6,4,3,1,2,5,7,9; 0,0,0,0,0,0,0,0,0; 0,10,0,0,0,0,0,11,0; 12,0,14,0,0,0,13,0,15; 0,0,0,0,0,0,0,0,0; 0,0,0,0,16,0,0,0,0; 32,0,30,0,31,0,29,0,28; 0,27,0,0,0,0,0,26,0; 0,0,0,0,0,0,0,0,0; 25,23,21,19,17,18,20,22,24]. It is the Red Player's turn.
Expected Behavior	The move is not allowed since Zu cannot move backward and nothing happens.
Test Result	Passed

4. Model Checking

This section provides an abstract model built in UPPAAL for model checking purposes. You can find the source files in tests/modelChecking and run it locally using an UPPAAL application (version>=4.1.26).

4.1 Introduction

The Chinese Chess system is divided into three components: controller, player red and

player black. Since the rule of Chinese Chess has been standardized and validated by the General Administration of Sport of China, our model focusses more on the integration level and less on the detailed logistics of Chinese Chess rules.

4.2 Assumptions

According to Chinese Chess rules, a game can last infinitely if players keep doing meaningless actions [1] and then lead to a deadlock. Since this kind of deadlock is considered valid in the rules, but will affect our validation for invalid deadlocks, some prevention measures are used in our validation model to avoid these valid deadlocks and allow continuous simulation, which may make it slightly different to the development model. These measures are based on the following facts.

- We assume both players are rational and will always choose the best strategy in a certain situation and hence none of them will do meaningless actions.
- For rational players, a game of chess can be finished in finite rounds, ranging from 3 to 1860 [2].

[1] Meaningless actions are defined as actions with no benefit to the situation. For example, move one piece of chess a step forward and in the next round move it a step backward, with no effect for either avoiding to loss it or bringing loss to the enemy.

[2] This data comes from the Internet.

4.3 Chess Game model

4.3.1 Red Player

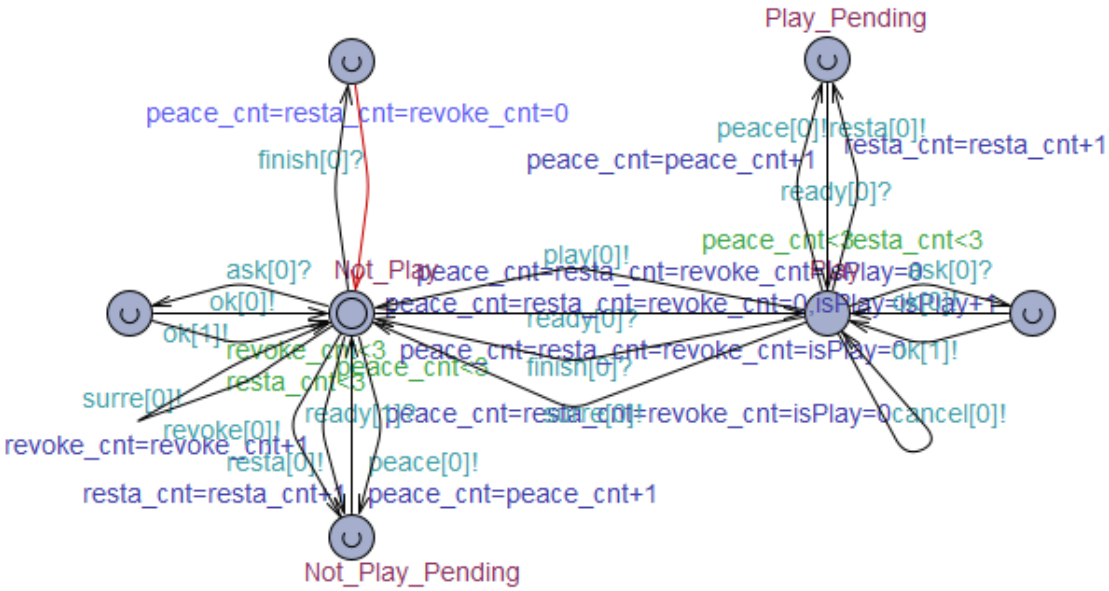

Fig 4.1. Red Player

Red Player has two major states: Play and Not Play.
In the Play state, Red Player can play the game, surrender, ask for peace or ask for restart, which will transfer it into Not Play state. Ask for restart and ask for peace can be only invoked 3 times at maximum in a round. In addition, Red Player will also respond to requests from the Black player when it asks for peace or restart. When the game is over, Red Player will receive a finish signal and return to Not Play state. And ready signal will make no sense when Red Player is in Play state.
In the Not Play state, Red Player can also surrender, ask for peace and ask for restart. Ask for restart and ask for peace can be only invoked 3 times at maximum in a round, as they are in the Play state. In addition, Red Player will also respond to requests from the Black player when it asks for peace or restart. When the game is over, Red Player will receive a finish signal and reset counts for ask for restart and ask for peace. And ready signal will transfer Red Player to Play state. In addition, Red Player can ask for revoke, which will reset the state of controller to the last playing state, and it can only be asked for at maximum 3 times.

4.3.2 Black Player

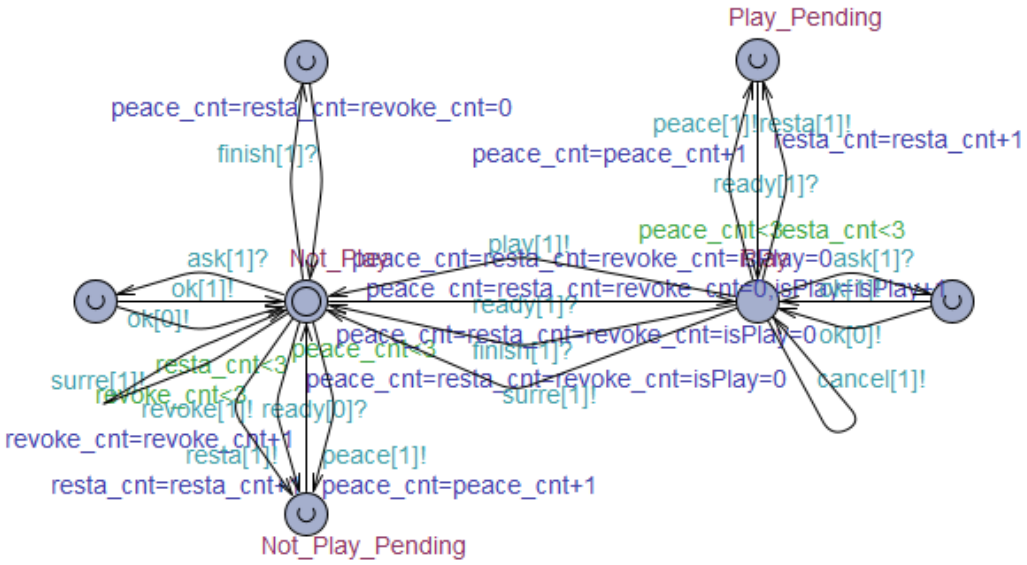

Fig 4.2. Black Player

The same as the Red Player, but only on the opposite channels. For convenience, we built the two players into two separate templates.

4.3.3 Controller

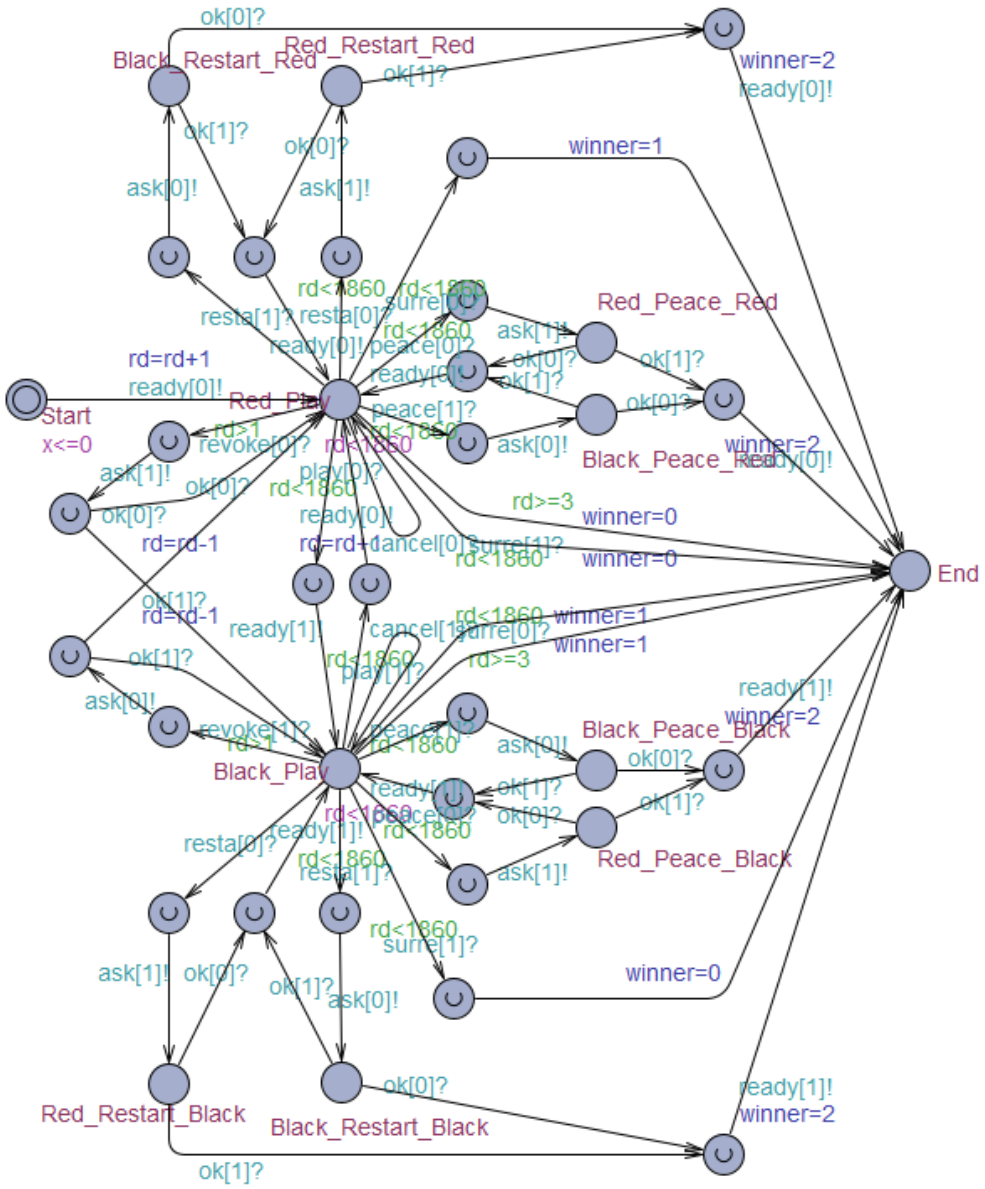

Fig 4.3. Controller

The controller has two four major states: Start, Red Play, Black Play and End.
In the Start state, the controller will leave it instantly the time it enters the state and enter Red Play state, as real-life chess starts with Red Play going first.
In the End state, the winner will be set symbolizing the end of the game, here we use 0 as Red Player wins, 1 as Black Player wins and 2 as no winner.
In the Red Play and Black Play state, the controller will leave it when any action is taken by the player or the opposite one. Among these actions, surrender will directly transfer the controller to the End state, while ask for peace and ask for restart require the

opposite player's decision. Play simulates real-life chess playing, based on the assumptions mentioned above, the controller can stay in the two states at maximum 1680 rounds as players can play at maximum 1680 rounds, and can transfer the controller into End state with a game winner after at least 3 rounds. Cancel will not cause any transitions and revoke will transfer the controller to its last state of playing if it is granted by the enemy.

4.3.4 Check Properties

4.3.4.1 Game Ending

Property	A<> controller.rd>0 imply controller.winner!=-1
Description	All executable paths that enter the game will eventually end.
Result	Passed

4.3.4.2 Play Once

Property	A[]red.isPlay<=1 && black.isPlay<=1
Description	At any time, no player can play twice in a row.
Result	Passed

4.3.4.3 Restart and Peace Limit

Property	A[]red.resta_cnt<=3 && red.peace_cnt<=3 && black.resta_cnt<=3 && black.peace_cnt<=3 && red.revoke_cnt<=3 && black.revoke_cnt<=3
Description	At any time, no player can ask for peace, restart or revoke for more than 3 times.
Result	Passed

4.3.4.4 Round Limit

Property	A[]controller.rd<=1860
Description	At any time, no game will last for more than 1860 rounds.
Result	Passed