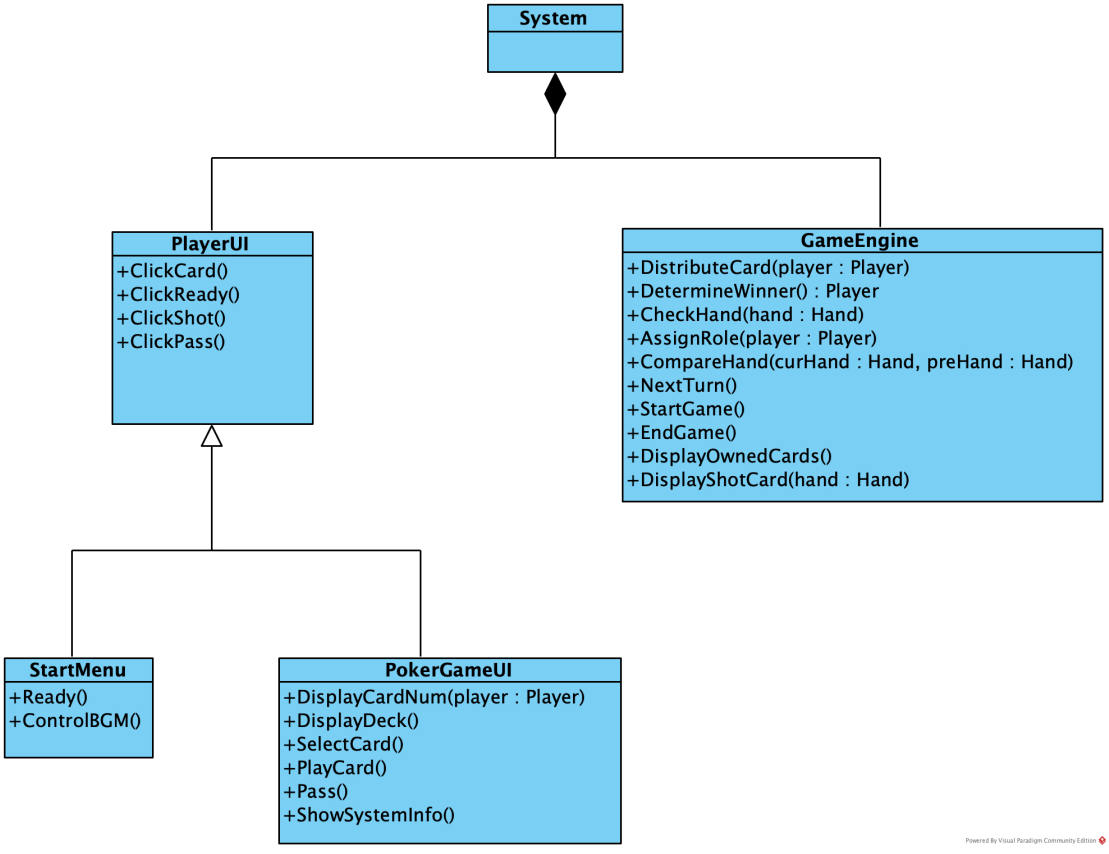
Validation for Poker Game

System Architecture



Functional Tests

For poker game, most of our functions can not operate individually. Thus, in this test, we should play the game to check if all the components that are linked to it and check if all the functions operate normally according to the rules. We will concentrate more on playing processes.

The processes are mainly validated by **branch coverage**. The cover number will be omitted if there is only one branch, as **Tcover1.1.2.1** shows.

Judging and comparing shotted cards by rules are the most important and complicated use case, and hard to be tested completely in the normal games (i.e. random roles and hand cards), thus they will be tested individually. Some other use cases are also hard to be tested when the hand cards are random, such as **T1.6**. They will be tested during rules test.

Use cases

- *T1.1: StartGame()*
    - T1.1.1: The Ready button can work normally.
        - Tcover1.1.1.1: When some but not all players in the room have clicked Ready button, the game engine should not start the game and wait. On each UI, its own player can see ready situation of others and blank avatars. After click, the button should disappear.
        - Tcover1.1.1.2: When three players in the room all have clicked **Ready** button, the game engine should start the game automatically.
        - State: Waiting

        - Test coverage: 2/2 = 100%
        - Test result: 2 passed
    - T1.1.2: There should be one and only one landlord. (Tcover1.1.2.1)
        - State: PlayHand
        - Test coverage: 1/1 = 100%
        - Test result: 1 passed
    - T1.1.3: The landlord should have 20 hand cards which is **three more** than others.
        - State: PlayHand
        - Test coverage: 1/1 = 100%
        - Test result: 1 passed

- *T1.2: DisplayOwnedCards()*
    - T1.2.1: After starting the game, the three players will see their own hand cards in their onw UI.
        - State: PlayHand
        - Test coverage: 1/1 = 100%
        - Test result: 1 passed
    - T1.2.2: The amount and the types of all hand cards should be a **complete** deck of poker.
        - State: PlayHand
        - Test coverage: 1/1 = 100%

- Test result: 1 passed

- *T1.3: Player.cardNum*:
  - T1.3.1: In each UI, its own player can see those infomation of others.
    - State: PlayHand
    - Test coverage: 1/1 = 100%
    - Test result: 1 passed

- *T1.4:* First player
  - T1.4.1: The landlord take **first** to shot the hand cards.
    - State: PlayHand
    - Test coverage: 1/1 = 100%
    - Test result: 1 passed

- *T1.5: whoseTurn* (Tested during rules test)
  - T1.5.1: Check if the game engine decides whose **turn** correctly.

```
testCase.press(player_0_UI.ShotButton);
testCase.verifyEqual(ge.whoseTurn, 1); %Tcover1.5.1.1
testCase.press(player_1_UI.PassButton);
testCase.verifyEqual(ge.whoseTurn, 1); %Tcover1.5.1.2
testCase.press(player_2_UI.PassButton);
```

```
testCase.press(player_2_UI.ShotButton);

testCase.verifyEqual(ge.whoseTurn, 0); %Tcover1.5.1.3
```

- Tcover1.5.1.1: player_0 to player_1
- Tcover1.5.1.2: player_1 to player_0
- Tcover1.5.1.3: player_2 to player_0
- State: PlayHand
- Test coverage: 3/3 = 100%
- Test result: 3 passed

- *T1.6: PlayHand()* (Tested during rules test)
  - *T1.6.1: select()*

```
[x,y] = ge.calcPosition(1,player_1);
testCase.press(player_1.currUI.UIFigure,[x,y]);

testCase.verifyEqual(player_1_UI.currPlayer.cards_img{1,1}.Position(1,2),52); %Tcover1.6.1.1
testCase.press(player_1.currUI.UIFigure,[x,y]);
testCase.verifyEqual(player_1_UI.currPlayer.cards_img{1,1}.Position(1,2),32); %Tcover1.6.1.2
```

- Tcover1.6.1.1: When *select()* the hand card, the card should pump up.
- Tcover1.6.1.2: When *select()* the hand card that has been selected, the card should go back to the original position.
- State: PlayHand
- Test coverage: 2/2 = 100%
- Test result: 2 passed

  - T1.6.2: *shot()*

```
n = [9,10,11,12,13,14,15,16,17];
for k = 1:length(n)
    [x,y] = ge.calcPosition(n(k),player_0);
    testCase.press(player_0.currUI.UIFigure,[x,y]);
end
testCase.verifyThat(@()testCase.press(player_0_UI.ShotButton),Throws(''));

testCase.verifyEqual(player_0_UI.UnknownTypeLabel.Text, 'Unknown Type!');
testCase.verifyEqual(player_0_UI.UnknownTypeLabel.Visible, 'on'); %Tcover1.6.2.1

n = [9];
for k = 1:length(n)
    [x,y] = ge.calcPosition(n(k),player_0);
    testCase.press(player_0.currUI.UIFigure,[x,y]);
end
testCase.press(player_0_UI.ShotButton);
testCase.verifyEqual(player_0_UI.UnknownTypeLabel.Text, 'Not Bigger!');
testCase.verifyEqual(player_0_UI.UnknownTypeLabel.Visible, 'on'); %Tcover1.6.2.2
```

- Tcover1.6.2.1: Select the cards whose **type is not valid**, click **出牌** button, the UI could show **unknown type error** infomation.
  - Cards chosed: KKKAAA10 10 8
- Tcover1.6.2.2: Select the cards that **is not bigger than the last shot** or **the type of which is not matched**, click **出牌** button, the UI could show **not bigger error** infomation.
  - Card chosed: 8

```
n = [1];
for k = 1:length(n)
    [x,y] = ge.calcPosition(n(k),player_1);
    testCase.press(player_1.currUI.UIFigure,[x,y]);
end
testCase.press(player_1_UI.ShotButton);

[nr, shotNum] = size(ge.cards_shotted_0);
mid = (shotNum + 1)/2;
for i = 1 : shotNum
    testCase.verifyEqual(player_0_UI.currDispCards{1, fix(10.5-(i-mid))}.ImageSource, ge.cards_shotted_0{3, shotNum - i + 1});
    testCase.verifyEqual(player_0_UI.currDispCards{1, fix(10.5-(i-mid))}.Visible, 'on');
    testCase.verifyEqual(player_1_UI.currDispCards{1, fix(10.5-(i-mid))}.ImageSource, ge.cards_shotted_0{3, shotNum - i + 1});
    testCase.verifyEqual(player_1_UI.currDispCards{1, fix(10.5-(i-mid))}.Visible, 'on');
    testCase.verifyEqual(player_2_UI.currDispCards{1, fix(10.5-(i-mid))}.ImageSource, ge.cards_shotted_0{3, shotNum - i + 1});
    testCase.verifyEqual(player_2_UI.currDispCards{1, fix(10.5-(i-mid))}.Visible, 'on');
end
testCase.verifyEqual(player_1_UI.CardNum_currplayer.Text,'4');
testCase.verifyEqual(player_0_UI.CardNum_player_1.Text,'4');
testCase.verifyEqual(player_2_UI.CardNum_player_2.Text,'4');
%Tcover1.6.2.3
```

- Tcover1.6.2.3: Select the cards that **valid**, click **出牌** button.

Then all three UIs will shows the **shotted cards**. The onwed cards and

num lable shoud decrease accordingly.

- Card chosed: 9, in the turn beginning

◆ State: PlayHand

◆ Test coverage: 3/3 = 100%

◆ Test result: 3 passed

- *T1.7: Pass()* (Tested during rules test)

```
% round 1
testCase.verifyEqual(player_1_UI.PassButton.Enable,'off'); %T1.7.1
```

■ T1.7.1: In the first turn, the landlord should not be able to push the **Pass button**.

◆ State: PlayHand

◆ Test coverage: 1/1 = 100%

◆ Test result: 1 passed

```
testCase.press(player_0_UI.PassButton); %T1.7.2
testCase.press(player_1_UI.PassButton);

% round 2
testCase.verifyEqual(player_2_UI.PassButton.Enable,'off'); %T1.7.3
```

■ T1.7.2: When clicking the **不出** button, the game engine will pass this turn;

◆ State: PlayHand

◆ Test coverage: 1/1 = 100%

◆ Test result: 1 passed

■ T1.7.3: Try **Pass** two turns, the third player should not be able to

push the **Pass button**.

- State: PlayHand
- Test coverage: 1/1 = 100%
- Test result: 1 passed

- *T1.8: EndGame()* (Tested during rules test)

```
testCase.verifyEqual(ge.isEnd, true); %T1.8.1
testCase.verifyEqual(ge.winner, 0); %Tcover1.8.2.1
```

  - T1.8.1: As soon as one of three playes' hand cards have **all shotted**, the game should end.
    - State: GameOver
    - Test coverage: 1/1 = 100%
    - Test result: 1 passed

```
testCase.verifyEqual(ge.winner, 1); %Tcover1.8.2.2
```

  - T1.8.2: The system should decide whether landlord wins correctly.
    - Tcover1.8.2.1: Landlord wins.
    - Tcover1.8.2.2: Peasants win.
    - State: GameOver
    - Test coverage: 2/2 = 100%
    - Test result: 2 passed

Rules

For test process details, see:

斗地主 demo 出牌.pdf

If the link is invalid, you can find the file in the work directory.

- Coverage items:
  - Tcover2.1.1: 三带一 bigger
  - Tcover2.1.2: 飞机带一 bigger
  - Tcover2.1.3: 顺子 bigger
  - Tcover2.1.4: 单张 bigger
  - Tcover2.1.5: 对子 bigger
  - Tcover2.1.6: 炸弹 normal
  - Tcover2.1.7: 连对 bigger
  - Tcover2.1.8: 王炸
  - Tcover2.2.1: Unknown type
  - Tcover2.3.1: Turn beginning
- Items should be covered:
  - 单张、顺子、对子、连对、三张、飞机不带、三带一、飞机带一、三带二、飞机带二、四带二、四带对子
    - bigger
    - not bigger
    - unmatched type
  - 炸弹

        - normal
        - bigger
        - not bigger
    - 王炸
    - Unknown type
    - Turn beginning
    - **In total: 12*3 + 3 + 1 + 1 + 1 = 42**
- Test coverage: 10/42 = 23.8%
- Test result: 2 tests passed, 10 coverage items passed