2018

# CSD2221: GANDERPOKE GAME

CSD2221: MINI PROJECT

KRISTEN REBELLO
M00572750

MIDDLESEX UNIVERSITY LONDON

# Table of Contents

# List of Figures

# Chapter 1: Requirements

## 1.1 Context

The card game, called 'Ganderpoke', is an original game by David Parlett which uses a traditional 52 card pack. The aim of this game is for a player to produce either the highest or the lowest five-card Poker combination from their hand of cards.

There are three and four player variations, but this particular version is a two-player game for two players sitting side-by-side at a single PC or laptop with a single keyboard and take turns using the same mouse or keyboard.

## 1.2 Domain Statement

Start with a traditional 52-card pack, add one joker and remove all twos, threes and fours. The pack will then contain 41 cards. Deal 25 cards face down in the center in a square of five rows and five columns. Divide the remaining 16 cards equally between the two players.

The non-dealer begins, and play continues in turns. At each turn, a player may take any face-down card from the square, add it to their hand, and fill the gap with any card from their hand, face-up. This may also be the card they have just picked up.

The Joker card has a different set of rules. If a player holds the Joker in their hand, they may take any card facing up from the square and leave the Joker face-up in its place. This ends the player's turn. If the Joker is lying face-up on the square, on their turn, the player may add it to their hand and fill the space with any card from their hand face-down.

Play ceases the moment all 25 cards of the square are face-up. The value of the square is calculated. The scores for the ten Poker combinations (five rows and five columns) in the square are calculated and scores 1 for a pair, 2 for two pairs, 3 for a triplet, 5 for a straight, flush or full-house, 8 for four-of-a-kind and 10 for a straight-flush. Those are summed up to provide a total score for the 5x5 square.

Whichever player can produce the highest Poker combination by using any five cards from their hand, scores twice the total score of the square. Similarly, the player who forms the lowest Poker combination, scores once the value of the square.

If a player holds the Joker at the end of play, then the player loses.

This completes one round of play. Players can decide how many rounds to play when they start.

The possibility does exist that both the players hold the same set of cards in their hands and both players score the highest and lowest possible Poker combinations. If both the players hold the same score on the last round, and this is the situation, the game ends in a draw or tie, unless the players decide to settle the score with another round.

## 1.3 Requirements Overview

### 1.3.1 Product Overview

Hypothetically, the aim is to develop a computer version of the Ganderpoke game described above. The intention is that, the game should be played by two players, sitting side-by-side, over a single PC or laptop. The game will let each player know when it is their turn and the other player must look away from the screen, while the first player is playing.

## 1.4 Workflow Models

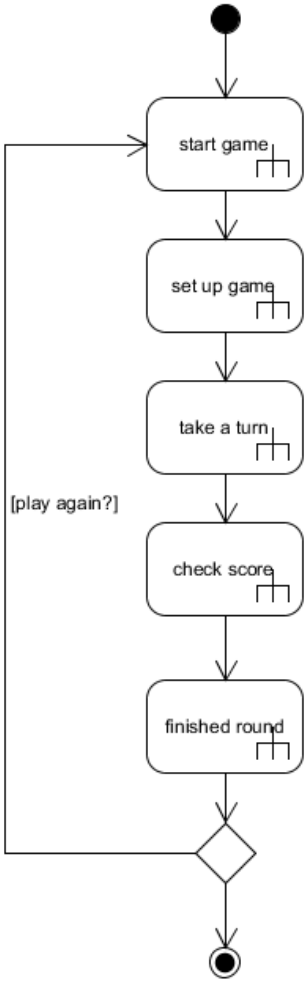


*Figure 1: Workflow Model - Main workflow model*

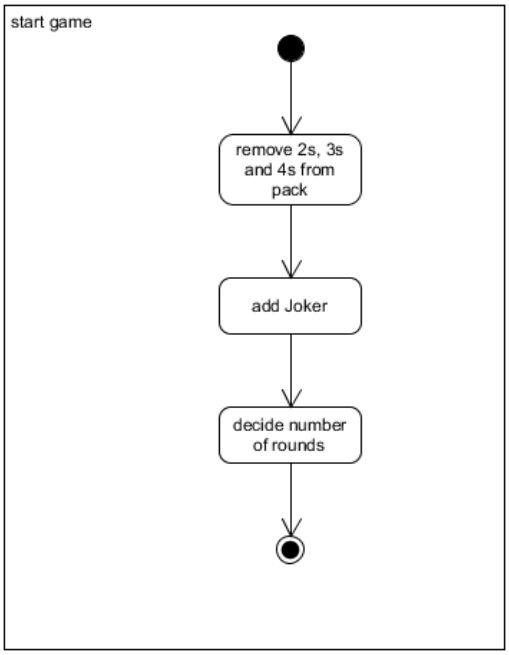


*Figure 2: Workflow Model - start game*

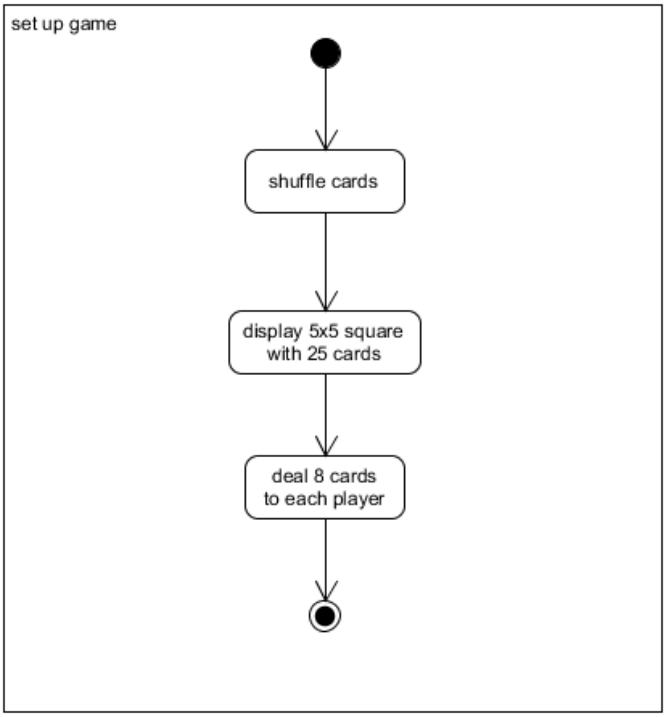


*Figure 3: Workflow Model - set up game*

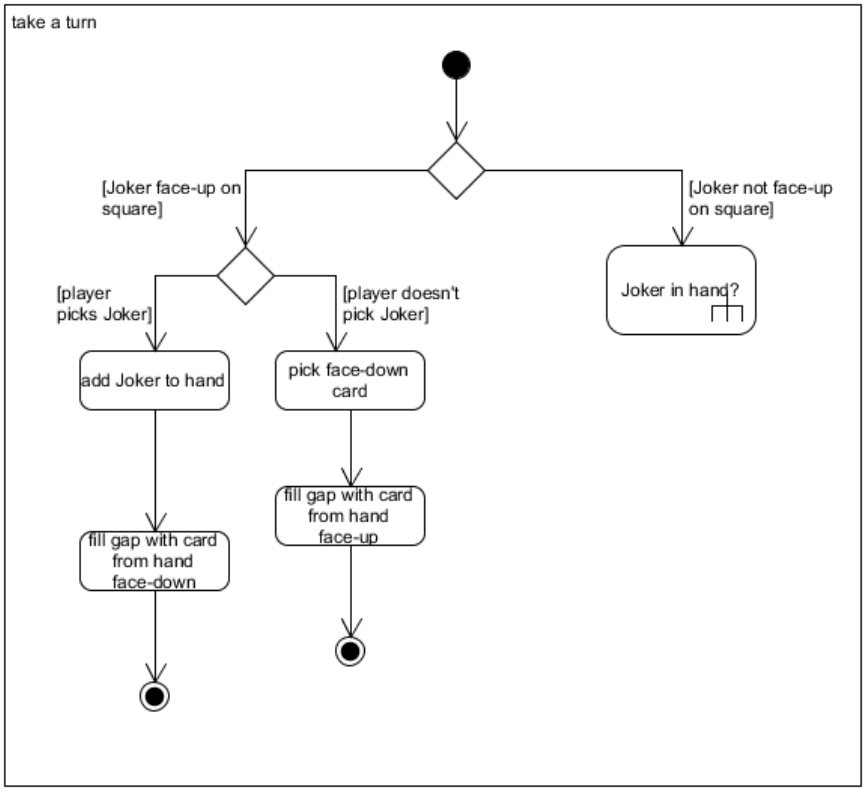


*Figure 4: Workflow Model - take a turn*

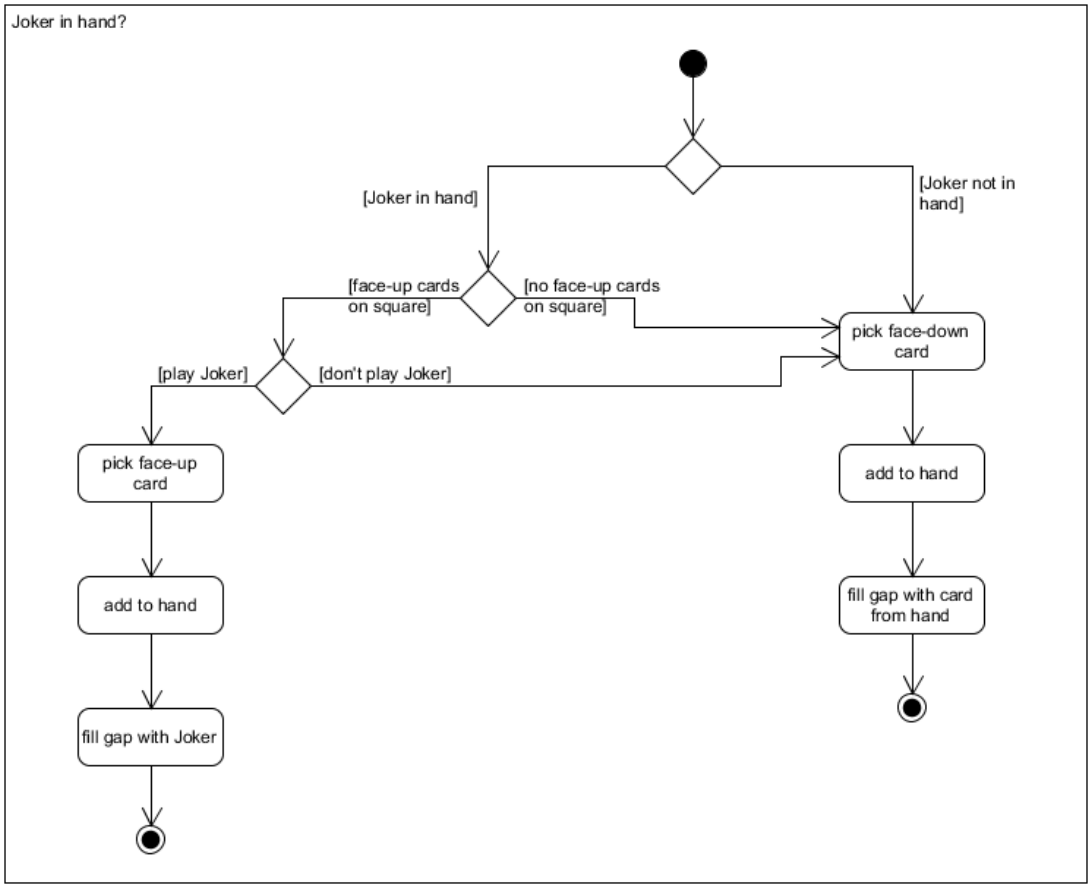


*Figure 5: Workflow Model - Joker in hand?*

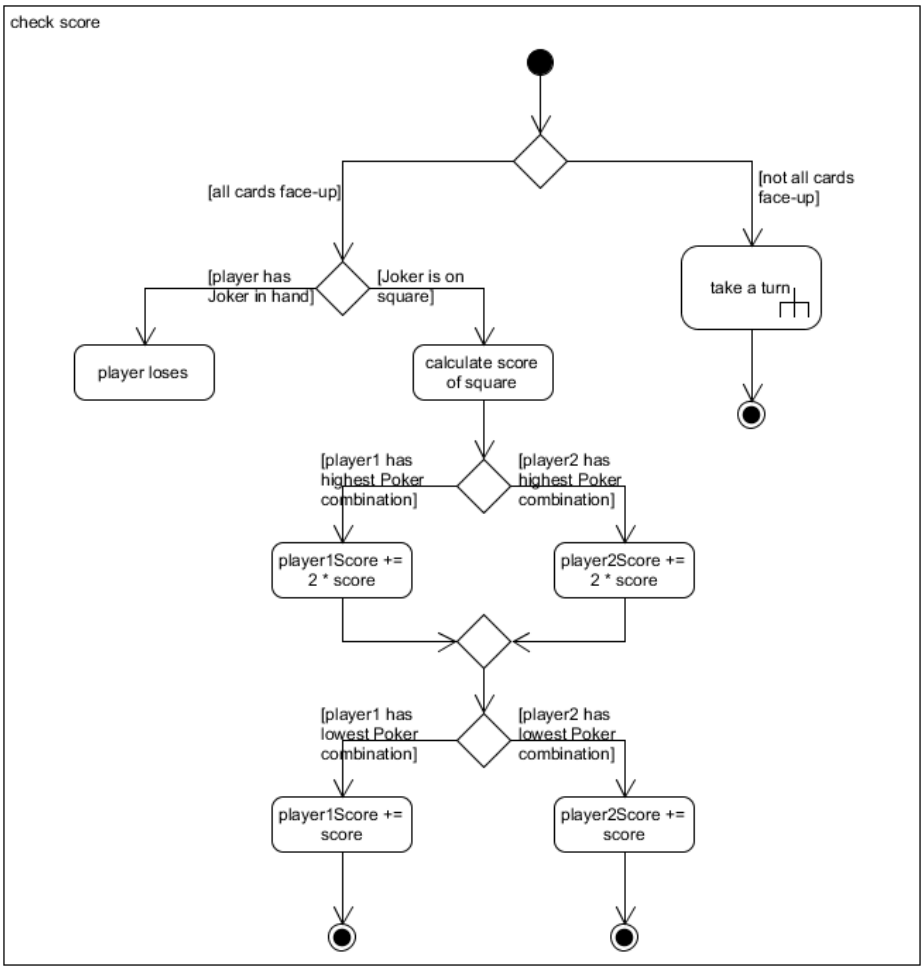


*Figure 6: Workflow Model - check score*

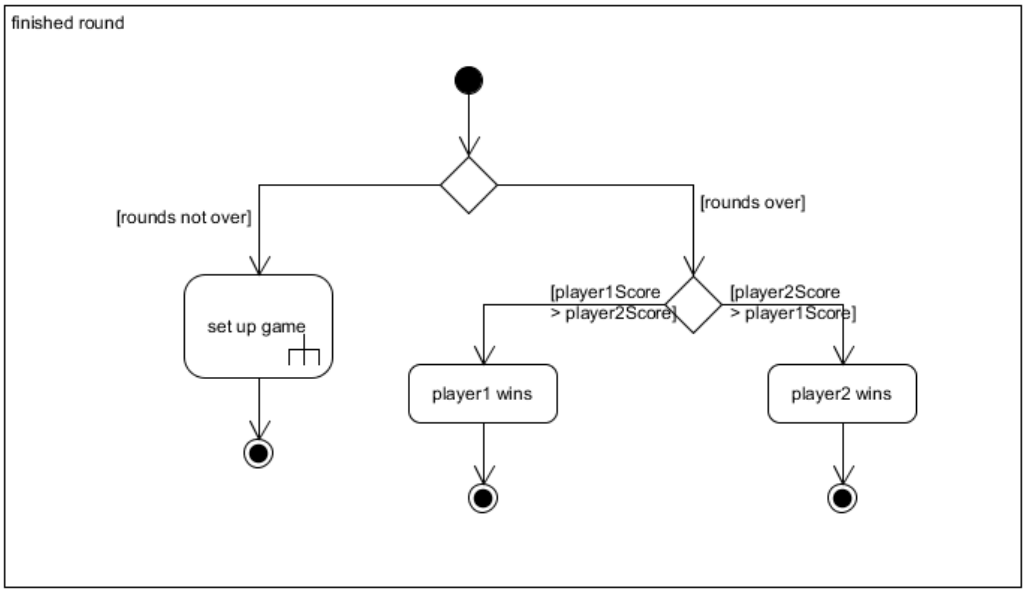


*Figure 7: Workflow Model - finished round*

## 1.5 Requirements

### 1.5.1 Functional Requirements

From the perspective of the game server: -

- Initiate a game of Ganderpoke
- Commence a game
    - Ask for number of rounds
    - Record number of rounds the player has entered
    - Shuffle pack
    - Set up the 5x5 square
    - Deal remaining cards between players
- Record a player move (i.e., pick up card)
- Check that a move is legal according to the game rules
- Give feedback of current game status to all players, e.g., indicating turns, whether to pick up cards etc.
- Calculate value of square
- Calculate score of players
- Check if all rounds are completed
- Compare scores of both players
- Announce the winner
- Reset to play another game

From the perspective of a client (player):-

- Click play/start
- Enter number of rounds
- Play the game
    - Pick up face down card from square
    - Add the card to player's hand
    - Remove card from player's hand
    - Place card face-up onto the square in empty place
- Give next player to play
- Check winner
- Exit the game

Obviously, as the design of the system starts to take shape, in terms of more refined implementation details, then the lower-level functional requirements can be derived, refined and documented.

### 1.5.2 Quality Requirements

- The target development language is Java 8, Standard Edition
- This should operate in the Netbeans 8 IDE within a Microsoft Windows environment
- The game should be designed to work for two
- A single PC or laptop with single mouse and/or keyboard should be used between two players

## 1.6 Use Cases

### 1.6.1 Use case: Start a game of Ganderpoke

| Identifier and name | | UC01: Start a game of Ganderpoke |
|---|---|---|
| Initiator | | Player1 |
| Goal | | Start a game |
| Start and stop points | | Player1 opens game, and Player1 is asked to play |
| Main success scenario | | |
| 1 | Player1 opens game | |
| 2 | The game displays start button | |
| 3 | Player1 clicks start | |
| 4 | System asks to enter number of rounds | |
| 5 | Player enters number of rounds | |
| 6 | The system(console) shuffles the pack of cards | |
| 7 | The system sets up the 5x5 square | |
| 8 | The system deals cards to each player | |
| 9 | Player2 is asked to turn away | |
| 10 | Player1 is asked to play | |

### 1.6.2 Use case: Take a turn

| Identifier and name | | UC02: Take a turn |
|---|---|---|
| Initiator | | A player |
| Goal | | Take a turn |
| Start and stop points | | A player clicks card, The player ends turn |
| Main success scenario | | |
| 1 | A player clicks card from square | |
| 2 | The system adds card to the player's hand | |
| 3 | The player selects card to replace from hand | |
| 4 | The player confirms card selection | |
| 5 | The system fills the empty space with the player's card | |
| 6 | The player ends turn | |
| Extensions | | |
| 1a | The player clicks face-up card and system prompts player to pick a face-down card, if no Joker in hand | |
| 1b | The player clicks face-up card and system prompts player to pick Joker card, if in hand | |
| 1c | The player clicks face-up Joker and system places replacement card face-down | |
| 5a | If Joker is selected, card is placed face-up | |
| 5b | If Joker was picked up, card selected is placed face-down | |
| 5c | Normal card selected, card is placed face-up | |

### 1.6.3 Use case: Check for winner

| Identifier and name | | UC03: Check for winner |
|---|---|---|
| Initiator | | System |
| Goal | | Check who's the winner |
| Start and stop points | | System checks for end, System announces winner |
| Main success scenario | | |
| 1 | A player completes a turn (Inclusion in UC02) | |
| 2 | A check is made regarding the state of the cards in the square | |
| 3 | If all the cards are face-up, the game is over | |
| 4 | The system locates Poker combinations in all rows and columns | |

| | |
|---|---|
| 5 | The value is then calculated and summed |
| 6 | The system checks for highest and lowest possible Poker combination in players cards |
| 7 | Player with highest Poker combination scores twice the value of the square |
| 8 | Player with lowest Poker combination scores once the value of the square |
| 9 | The system checks if the round number matches the number of rounds selected |
| 10 | If rounds match, the system checks if either player has a Joker card |
| 11 | If no player has a Joker card, system compares scores between players |
| 12 | System displays winner and results |
| Extensions: | |
| 3a | If all cards are not face up, next player takes a turn |
| 10a | If rounds don’t match, system refreshes for a new round |
| 11a | If player holds a Joker card, player loses |
| 12a | If player scores are same, system displays draw |

# Chapter 2: Initial Conceptual Model

## 2.1 Class Diagram

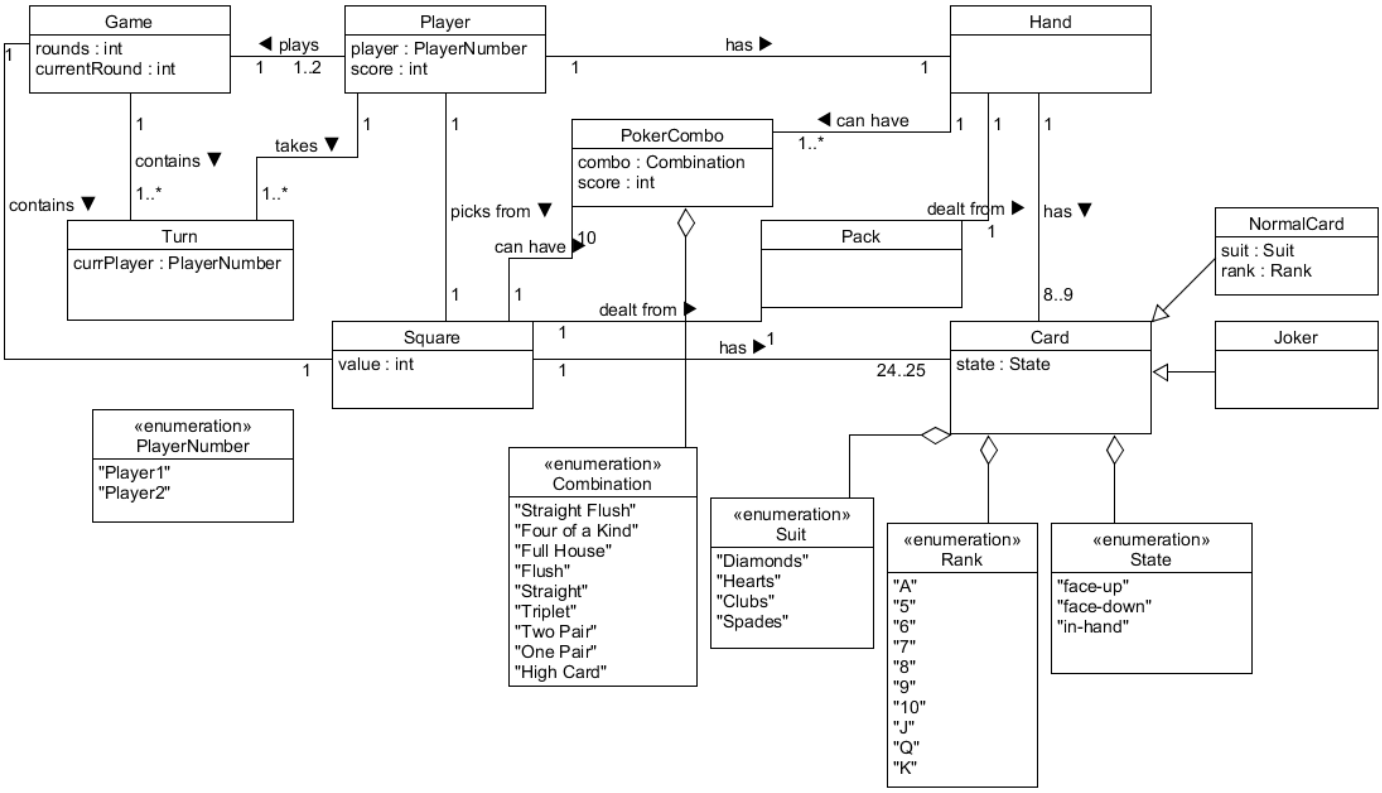


*Figure 8: Conceptual Model*

## 2.2 Class Descriptions

| | | |
|---|---|---|
| **Class** | `Card` | A card in the Ganderpoke game<br>Generalises `Joker` and `NormalCard` |
| | **Attributes** | |
| | `state` | The state of the card (face-up, face-down, in-hand) |
| **Class** | `Player` | A person who is playing the game |
| | **Attributes** | |
| | `player` | Player number (Player1, Player2) |
| | `score` | Score of the player |
| **Class** | `Game` | A game of Ganderpoke |
| | **Attributes** | |
| | `rounds` | The number of rounds selected by the player |
| | `currentRound` | The current round of play |
| **Class** | `NormalCard` | A normal card in the Ganderpoke game<br>Specialises `Card` |

| | | |
|---|---|---|
| | **Attributes** | |
| | `suit` | Suit of the card (Diamonds, Hearts, Clubs, Spades) |
| | `rank` | Rank of the card (A, 5, 6, 7, 8, 9, 10, J, Q, K) |
| **Class** | `Joker` | A Joker card in the Ganderpoke game<br>Specialises `Card` |
| | **Attributes** | |
| | `None` | |
| **Class** | `PokerCombo` | A Poker combination |
| | **Attributes** | |
| | `combo` | The type of combination (“Straight Flush”, “Four of a Kind”, “Full House”, “Flush”, “Straight”, “Triplet”, “Two Pair”, “One Pair”, “High Card”) |
| | `score` | The score awarded to the player |
| **Class** | `Square` | The 5x5 square displayed on the table/screen |
| | **Attributes** | |
| | `value` | Value of the square after the round has been completed |
| **Class** | `Hand` | A collection of cards currently held by a player |
| | **Attributes** | |
| | `None` | |
| **Class** | `Turn` | The event of a player taking a turn in the game |
| | **Attributes** | |
| | `player` | Current player taking a turn |
| **Class** | `Pack` | The pack of cards used in the Ganderpoke game |
| | **Attributes** | |
| | `None` | |

## 2.3 Invariants

I. If a `Player` object, `Player1`, is linked to a `Game` object, `aGame`, and the `Game` object, `aGame`, is linked to a `Turn` object, `aTurn`, then `Player1` must be linked to `aTurn`.

II. If a `Card` object, `aCard`, is linked to a `Square` object, `aSquare`, then the `state` in `aCard` must either be `"face-up"` or `"face-down"`.

III. If a `Card` object, `aCard`, is linked to a `Hand` object, `aHand` then the `state` in `aCard` must be `"in-hand"`.

IV. A `Square` object, `aSquare`, has an initial `value` of `0`, when linked to a `Game` object, `aGame`.

V. If a `PokerCombo` object, `aPokerCombo`, has a `combo` of `"Straight Flush"`, then the `score` in `aPokerCombo` should be `10`.

VI. If a `PokerCombo` object, `aPokerCombo`, has a `combo` of `"Four of a Kind"`, then the `score` in `aPokerCombo` should be `8`.

VII. If a `PokerCombo` object, `aPokerCombo`, has a `combo` of `"Full House"`, `"Flush"` or `"Straight"`, then the `score` in `aPokerCombo` should be `5`.

VIII. If a `PokerCombo` object, `aPokerCombo`, has a `combo` of `"Triplet"`, then the `score` in `aPokerCombo` should be `3`.

IX. If a `PokerCombo` object, `aPokerCombo`, has a `combo` of `"Two Pair"`, then the `score` in `aPokerCombo` should be `2`.

X. If a `PokerCombo` object, `aPokerCombo`, has a `combo` of `"One Pair"`, then the `score` in `aPokerCombo` should be `1`.

XI. If a `PokerCombo` object, `aPokerCombo`, has a `combo` of `"High Card"`, then the `score` in `aPokerCombo` should be `0`.

# Chapter 3: Dynamic Designs

## 3.1 Operation Specifications

### 3.1.1 Operation Specification: Pick Up Card

| | |
|---|---|
| Identifier | OS 01, version 1.0 |
| Context | Player |
| Signature | pickUpCard(Card : aCard) : void |
| Invariant | True |
| Precondition | |
| | The selected card is face-down, and it is the current player's turn |
| Postcondition | |
| | The card is added to the player's hand<br>The square contains one less card<br>The player's hand contains one more card<br>The player must select a card to replace |

### 3.1.2 Operation Specification: Is Joker In Hand?

| | |
|---|---|
| Identifier | OS 02, version 1.0 |
| Context | Hand |
| Signature | jokerInHand(Hand : aHand) : boolean |
| Invariant | True |
| Precondition | |
| | All cards are face-up and the round is over |
| Postcondition | |
| | If player's hand contains Joker, player has lost<br>If player's hand does not contain Joker, score is calculated<br>End of round |

### 3.1.3 Operation Specification: Place Card on Square

| | |
|---|---|
| Identifier | OS 03, version 1.0 |
| Context | Player |
| Signature | placeCard(Card : aCard) : void |
| Invariant | True |
| Precondition | |
| | The selected card is valid, and it is the current player's turn |
| Postcondition | |
| | The card is placed on the empty spot<br>The player's hand has one less card<br>The square has one more card<br>It is the next player's turn |

### 3.1.4 Operation Specification: Compare Score

| | |
|---|---|
| Identifier | OS 04, version 1.0 |
| Context | Game |
| Signature | compareScore(Player p1, Player p2) : Player |
| Invariant | True |
| Precondition | |
| | The number of rounds is over, and players have been assigned a score |
| Postcondition | |
| | A winner is announced<br>Game over |

### 3.1.5 Operation Specification: Is Card Valid?

| | |
|---|---|
| Identifier | OS 05, version 1.0 |
| Context | Game |
| Signature | isValid(Card card1, Card card2) : boolean |
| Invariant | True |
| Precondition | |
| | The current cards are aState, aSuit and aRank<br>card1 is connected to the square and card2 is linked to the player's hand |
| Postcondition | |
| | If aState of card1 is face-up and card2 is a Joker, return true<br>If aState of card1 is face-down and card2 is a normalCard, return true<br>If card1 is a Joker and aState of card2 is face-up, return true<br>Otherwise return false |

### 3.1.6 Operation Specification: Check For Number of Rounds

| | |
|---|---|
| Identifier | OS 06, version 1.0 |
| Context | Game |
| Signature | setCurrentRound() : void |
| Invariant | True |
| Precondition | |
| | The current round has just ended |
| Postcondition | |
| | If the currentRound < rounds, the current round has been incremented<br>Otherwise the game is over |

## 3.2 Sequence Diagrams

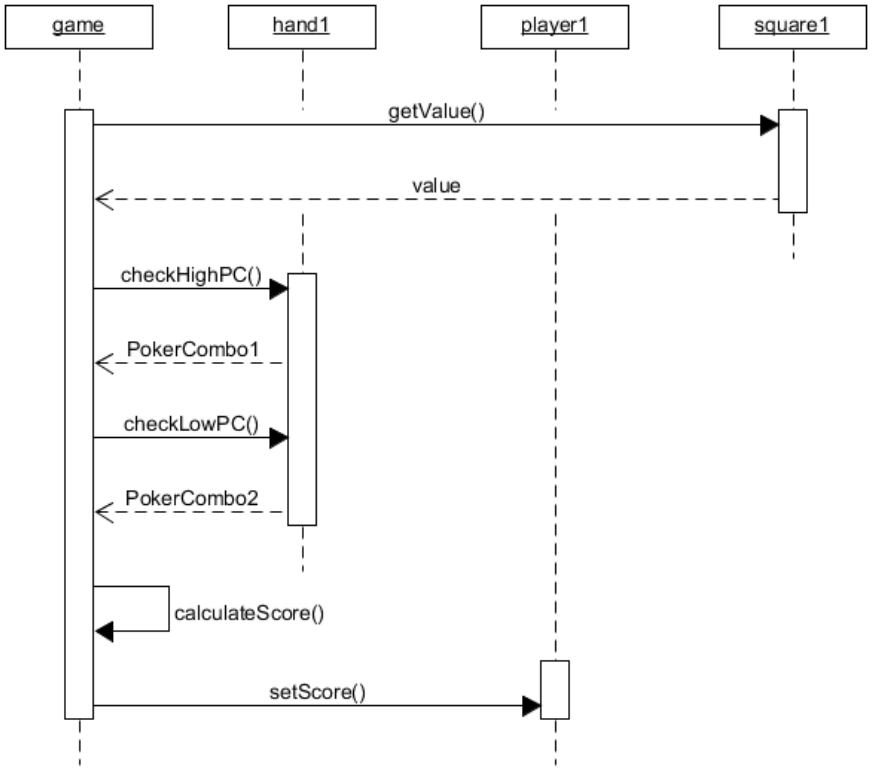


*Figure 9: Sequence Diagram - set score*

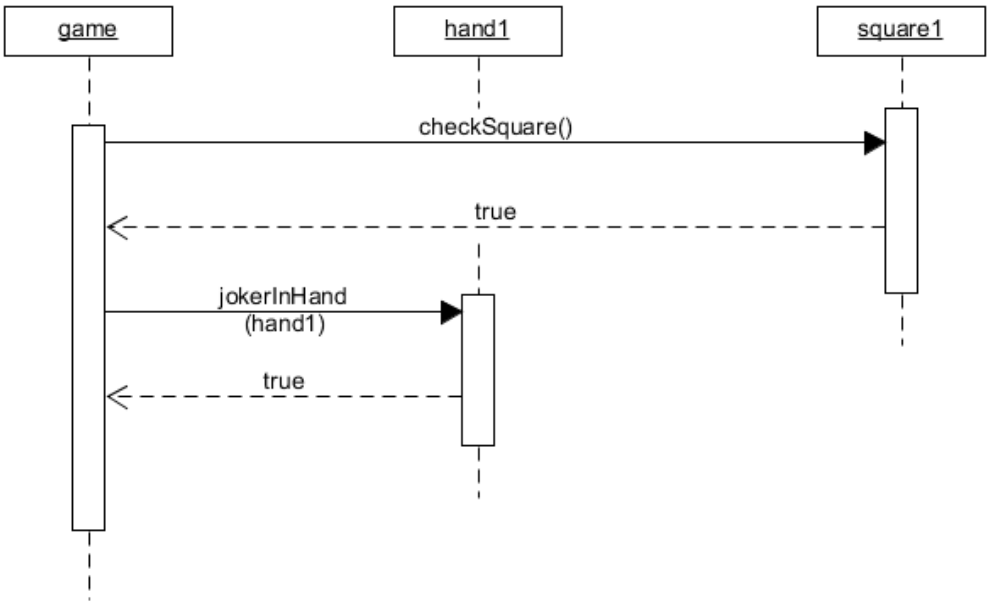


*Figure 10: Sequence Diagram - is Joker in hand?*

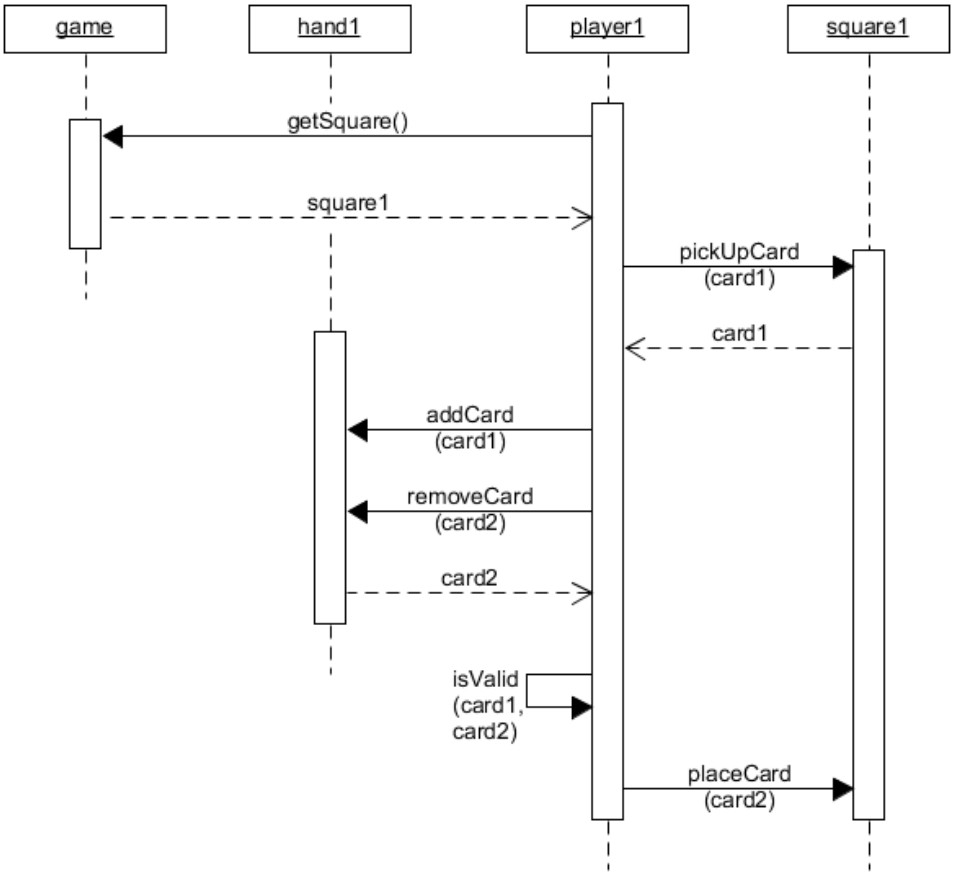


*Figure 11: Sequence Diagram - take a turn*

## 3.3 Object Diagrams

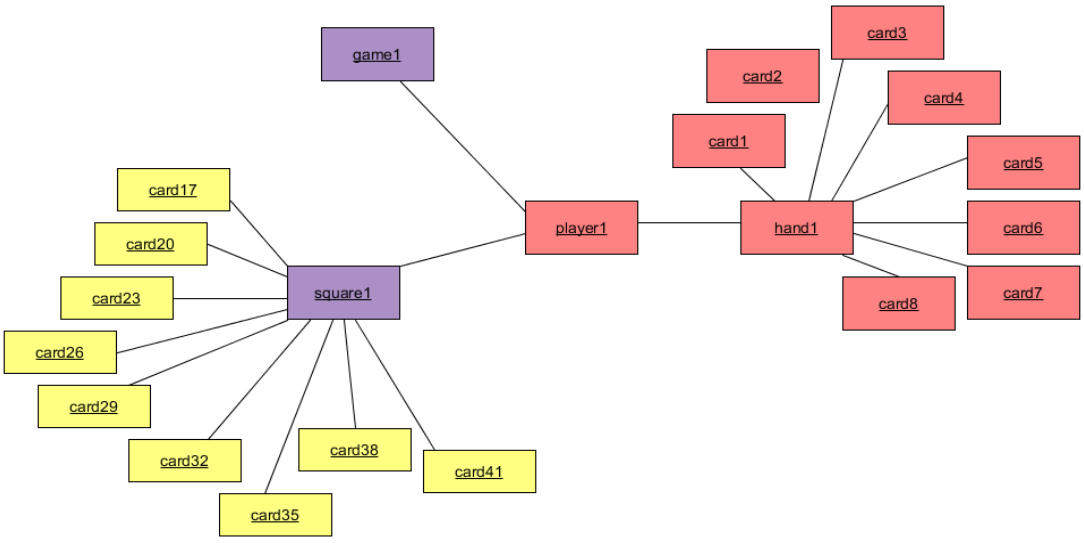


*Figure 12: Object Diagram - Pre-condition of pickUpCard(Card c)*

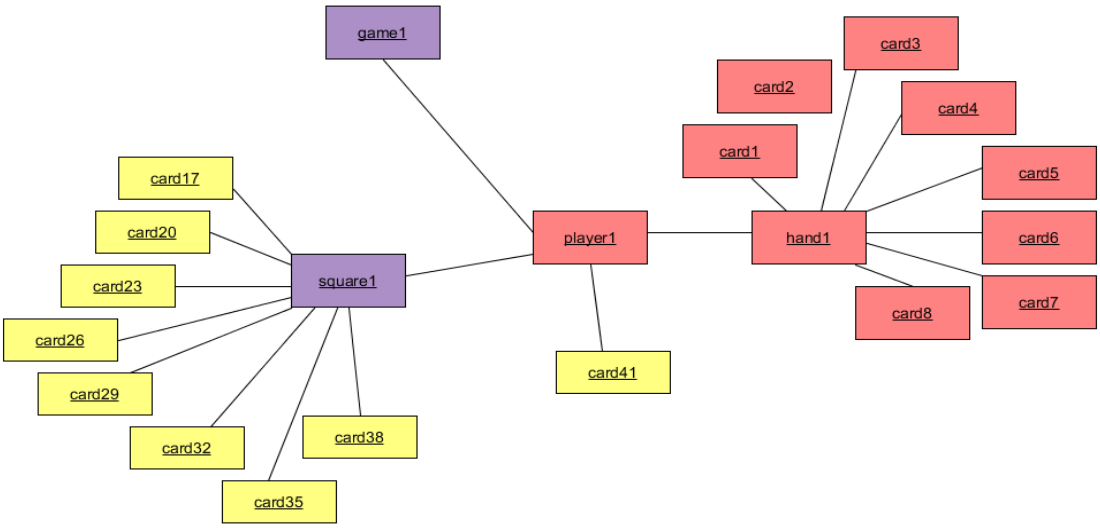


*Figure 13: Object Diagram - Post-condition of pickUpCard(card41) & Pre-condition of addCard(card41)*

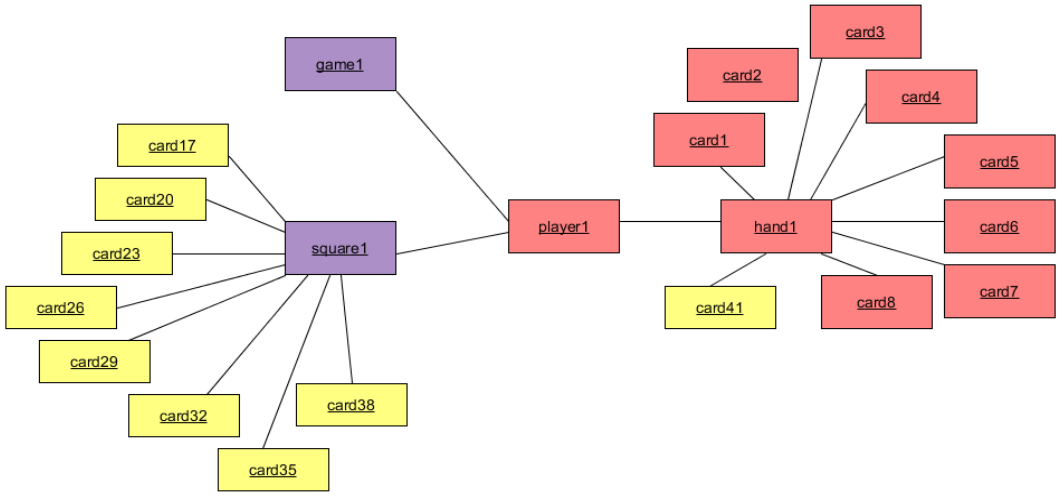


*Figure 14: Object Diagram - Post-condition of addCard(card41)*

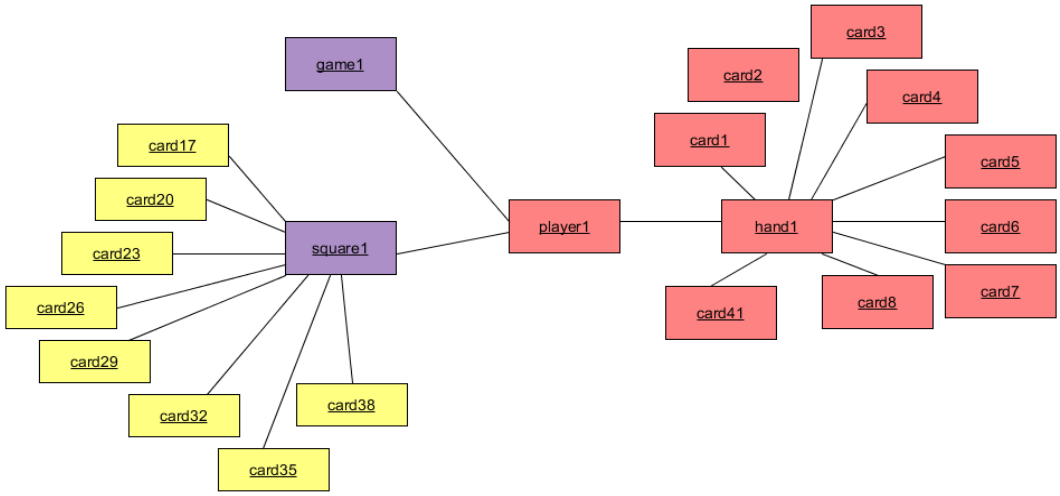


*Figure 15: Object Diagram - Pre-condition of removeCard(Card c)*

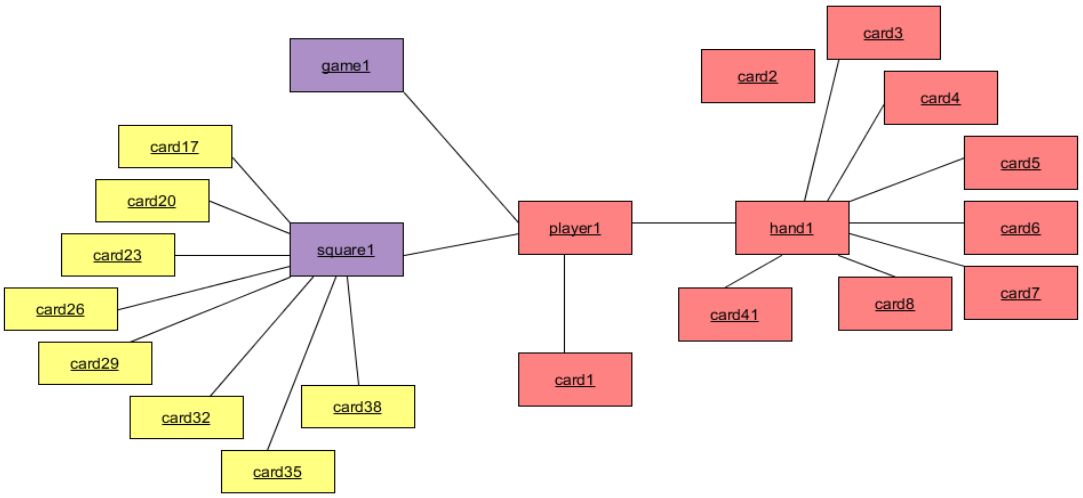


*Figure 16: Object Diagram - Post-condition of removeCard(card1) & Pre-condition of placeCard(card1)*

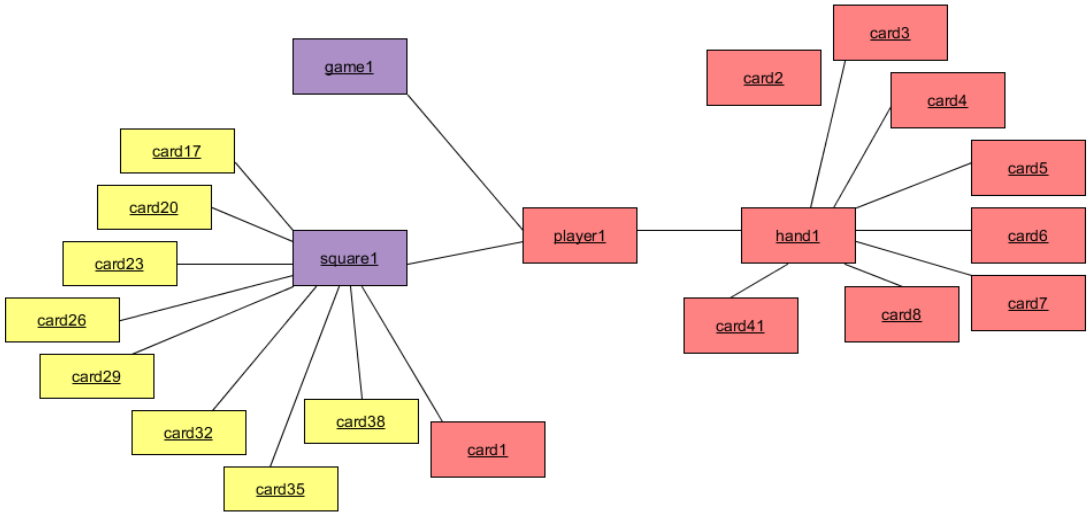


*Figure 17: Object Diagram - Post-condition of placeCard(card1)*

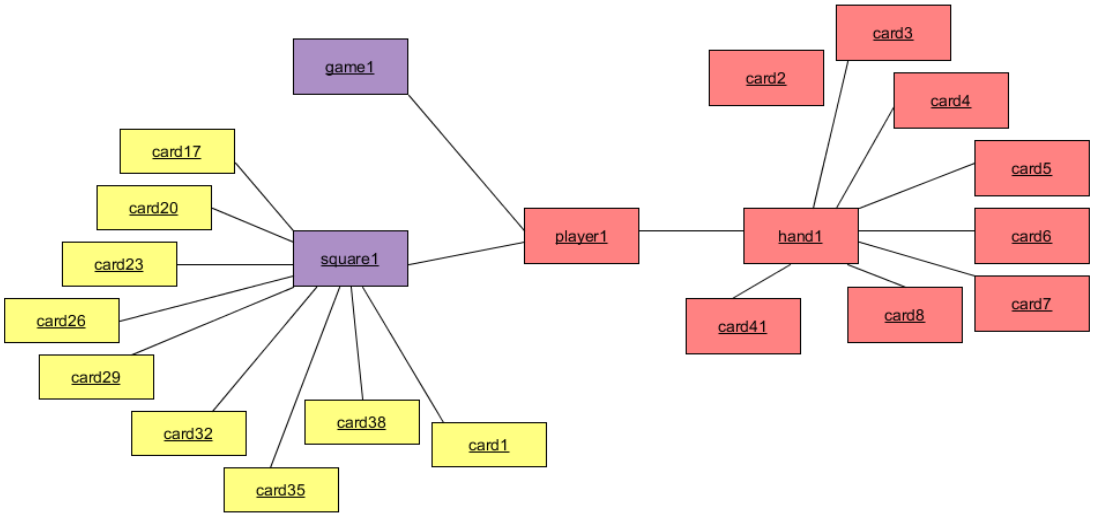


*Figure 18: Object Diagram - One turn over; next player's turn*

# 3.4 Implementation

## 3.4.1 Implementation Model

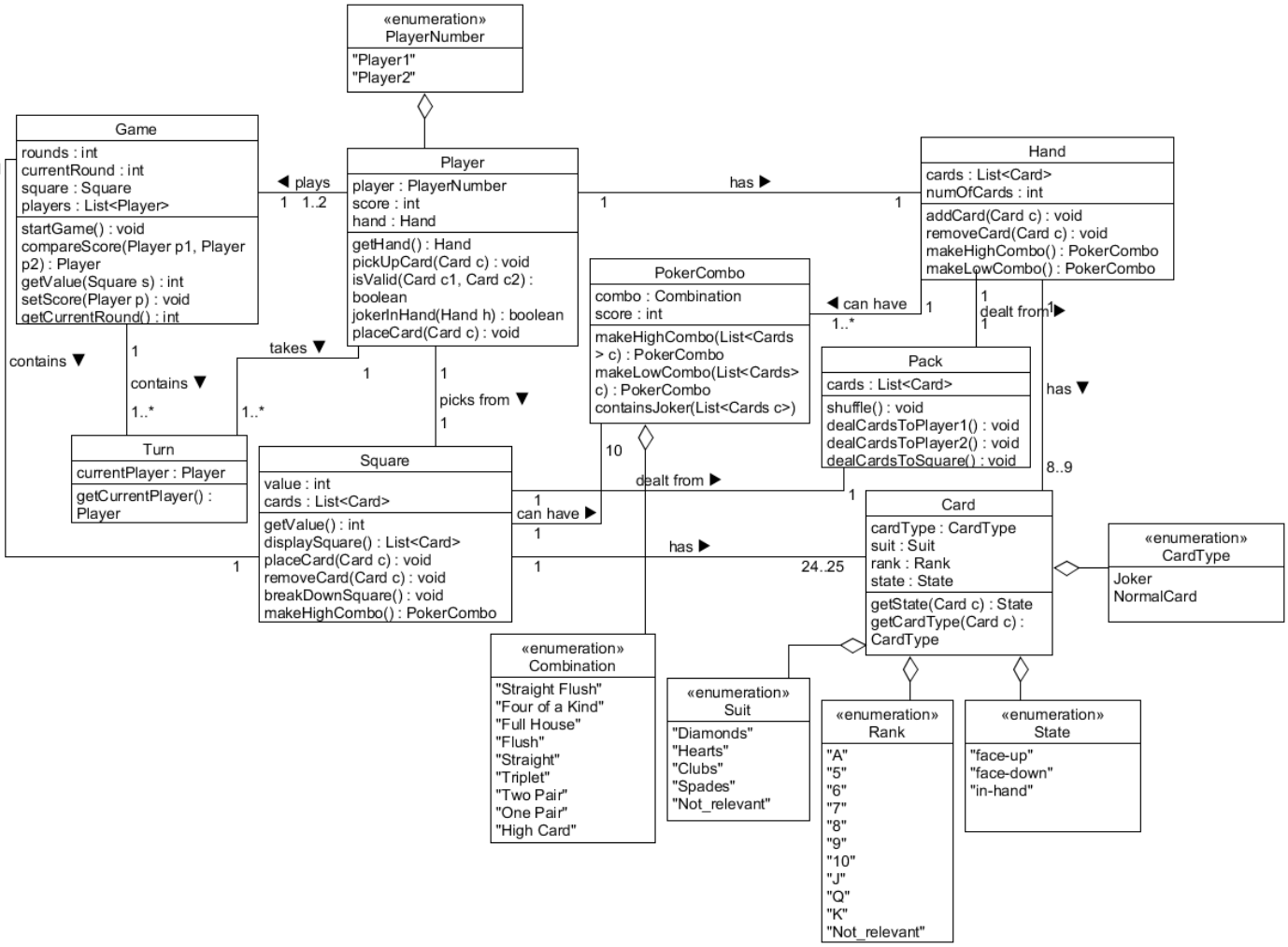


*Figure 19: Implementation Model*

## 3.4.2 Commentary

The methods of all the classes have been fitted into the above implementation model.

For the `Card` class, the specialisations (`Joker` and `NormalCard`) have been omitted and `cardType` has been introduced with the values restricted to being from the enumeration `CardType`.

The `Turn` class contains a link to a `Player` object which returns the current player (`currentPlayer`) who is playing their turn.

**Updated Invariants:**

- If a `Card` object, `aCard`, has a `cardType` of `Joker`, then the `suit` in `aCard` must be `"Not_relevant"`.
- If a `Card` object, `aCard`, has a `cardType` of `Joker`, then the `rank` in `aCard` must be `"Not_relevant"`.

**LINKS**

- the **Game** class holds a link reference to a **Square**
- the **Game** class holds a collection of **Player** objects
- the **Player** class holds a link reference to a **Hand**
- the **Square** class holds a collection of **Card** objects
- the **Hand** class holds a collection of **Card** objects
- the **Turn** class holds a link reference to a **Player**
- the **Pack** class holds a collection of **Card** objects

**METHODS**

- class **Turn** had method `getCurrentPlayer()`
- class **Game** has method `getCurrentRound()`
- class **Game** has method `compareScore(Player p1, Player p2)`
- class **Game** has method `getValue(Square s)`
- class **Game** has method `setScore(Player p)`
- class **Game** has method `startGame()`
- class **Player** has method `getHand()`
- class **Player** has method `pickUpCard(Card c)`
- class **Player** has method `placeCard(Card c)`
- class **Player** has method `isValid(Card c1, Card c2)`
- class **Player** has method `jokerInHand(Hand h)`
- class **Square** has method `placeCard(Card c)`
- class **Square** has method `pickCard(Card c)`
- class **Square** has method `displaySquare()`
- class **Square** has method `getValue()`
- class **Square** has method `makeHighCombo()`
- class **Square** has method `breakDownSquare()`
- class **Hand** has method `addCard(Card c)`
- class **Hand** has method `removeCard(Card c)`
- class **Hand** has method `makeHighCombo(Hand h)`
- class **Hand** has method `makeLowCombo(Hand h)`
- class **Card** has method `getState()`
- class **PokerCombo** has method `makeHighCombo(List<Card> c)`
- class **PokerCombo** has method `makeLowCombo(List<Card> c)`
- class **PokerCombo** has method `containsJoker(List<Card> c)`
- class **Pack** has method `shuffle()`
- class **Pack** has method `dealCardsToPlayer1()`
- class **Pack** has method `dealCardsToPlayer2()`
- class **Pack** has method `dealCardsToSquare()`

### 3.4.3 Updated Class Descriptions

**Class** `Card` — A card in the Ganderpoke game

**Attributes**

| | |
|---|---|
| `state` | The state of the card (face-up, face-down, in-hand) |
| `suit` | The suit of the card (“Diamonds”, “Hearts”, “Clubs”, “Spades”, “Not_relevant”) |
| `rank` | The rank of the card (“A”, “5”, “6”, “7”, “8”, “9”, “10”, “J”, “Q”, “K”, “Not_relevant”) |
| `cardType` | Type of card (Joker, NormalCard) |

**Protocol**

```
State getState(Card c)
CardType getCardType(Card c)
```

**Class** `Player` — A person who is playing the game

**Attributes**

| | |
|---|---|
| `player` | Player number (Player1, Player2) |
| `score` | Score of the player |

**Links**

| | |
|---|---|
| `Hand hand` | References the linked `Hand` object |

**Protocol**

```
Hand getHand()
void pickUpCard(Card c)
void placeCard(Card c)
boolean isValid(Card c1, Card c2)
boolean jokerInHand()
```

**Class** `Game` — A game of Ganderpoke

**Attributes**

| | |
|---|---|
| `rounds` | Number of rounds the player has entered |
| `currentRound` | The current round of play |

**Links**

| | |
|---|---|
| `Square square` | References the linked `Square` object |
| `Collection<Player> players` | References a collection of the linked `Player` objects |

**Protocol**

```
int getCurrentRound()
void startGame()
Player compareScore(Player p1, Player p2)
int getValue(Square s)
void setScore(Player p)
```

**Class** `PokerCombo` — A Poker combination

**Attributes**

| | |
|---|---|
| `combo` | The type of combination (“Straight Flush”, “Four of a Kind”, “Full House”, “Flush”, “Straight”, “Triplet”, “Two Pair”, “One Pair”, “High Card”) |
| `score` | The score awarded to the player |

**Links**

`None`

**Protocol**

```
PokerCombo makeHighCombo(List<Card> c)
PokerCombo makeLowCombo(List<Card> c)
boolean containsJoker(List<Card> c)
```

**Class** `Square` The 5x5 square displayed on the table/screen

**Attributes**

`value` Value of the square after the round has been completed

**Links**

`Collection<Card> cards`
References a collection of the linked `Card` objects

**Protocol**

```
void placeCard(Card c)
void removeCard(Card c)
void breakDownSquare()
int getValue()
List<Card> displaySquare
PokerCombo checkCombo(List<Card> l)
```

**Class** `Hand` A collection of cards currently held by a player

**Attributes**

`numOfCards` The number of cards in the hand

**Links**

`Collection<Card> cards`
References a collection of the linked `Card` objects

**Protocol**

```
void addCard(Card c)
void removeCard(Card c)
PokerCombo makeHighCombo()
PokerCombo makeLowCombo()
```

**Class** `Turn` The event of a player taking a turn in the game

**Attributes**

`None`

**Links**

`Player currentPlayer`
References the linked `Player` object

**Protocol**

```
Player getCurrentPlayer()
```

Class `Pack` The pack of cards in the Ganderpoke game

**Attributes**

`None`

**Links**

`List<Card> cards`
References a collection of the linked `Card` objects

**Protocol**

```
void shuffle()
Void dealCardsToPlayer1()
void dealCardsToPlayer2()
Void dealCardsToSquare()
```

# Chapter 4: User Interface Interaction

## 4.1 Mock System

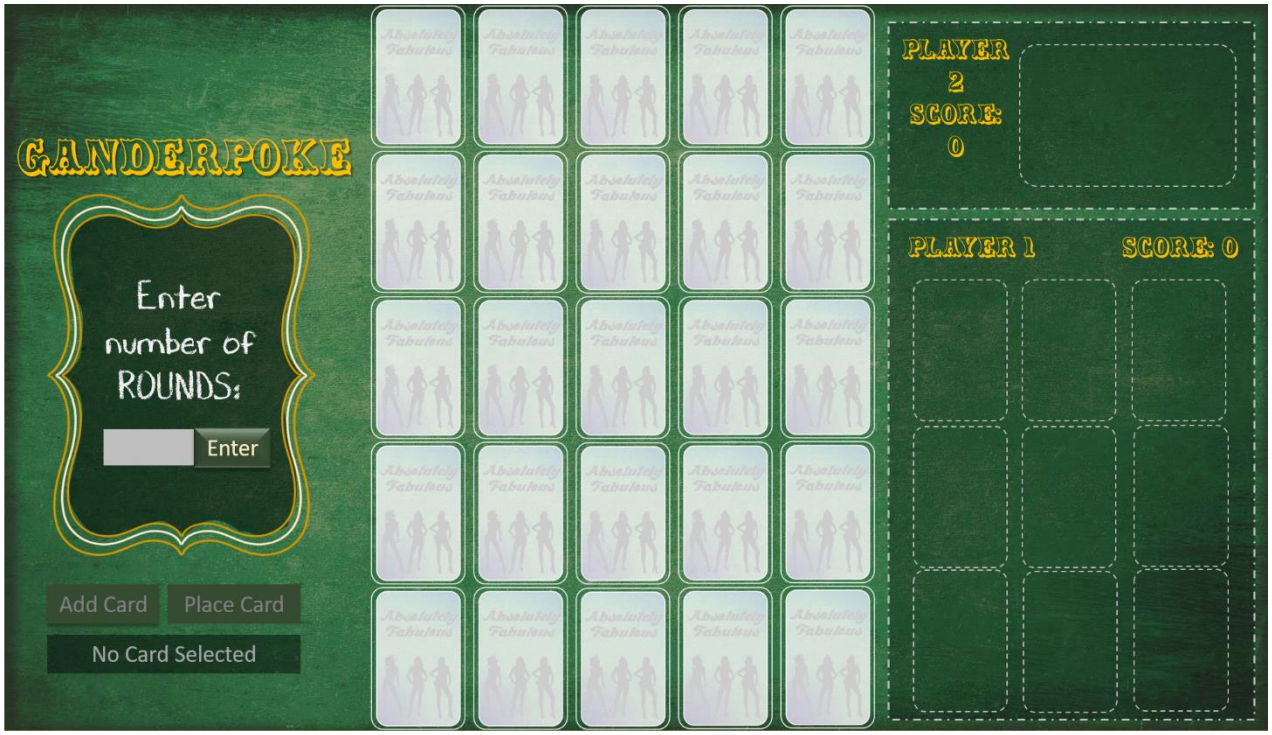


*Figure 20: Mock GUI of Ganderpoke game initial set up*


*Figure 21: Mock GUI of Ganderpoke game in play*


*Figure 22: Mock GUI of Ganderpoke game after player turn*

## 4.2 User Interface Design Commentary

A few mock ups of the game Ganderpoke GUI have been designed and included in this report. The visibility of the features seems to be a little tight fit and crowded, but overall allows the user to play and view all options.

Figure 20 shows an initial start of the game. Once a player clicks on the start button (not shown in mock ups), it will proceed to this screen and will prompt the user to enter the number of rounds that they would like to play. The rounds are then displayed in the upper left-hand corner, as shown in Figure 22.

In Figure 21 and 22, we can see that the user has buttons available to use. These have been given obvious names for the user's convenience and have been disabled when not allowed to use.

User feedback is provided on the left, in the frame, and messages to the user are simple, but well-defined and unambiguous. It tells the user what to do, in the case of Figure 22, it is giving player1 feedback that his/her turn is over, and they can hide their cards when ready and give to the next player.

It is simple, with the on-screen instructions, and in the actual game, there would be instructions on how to play the game as well (not shown in mock ups).

## 4.3 State Diagrams

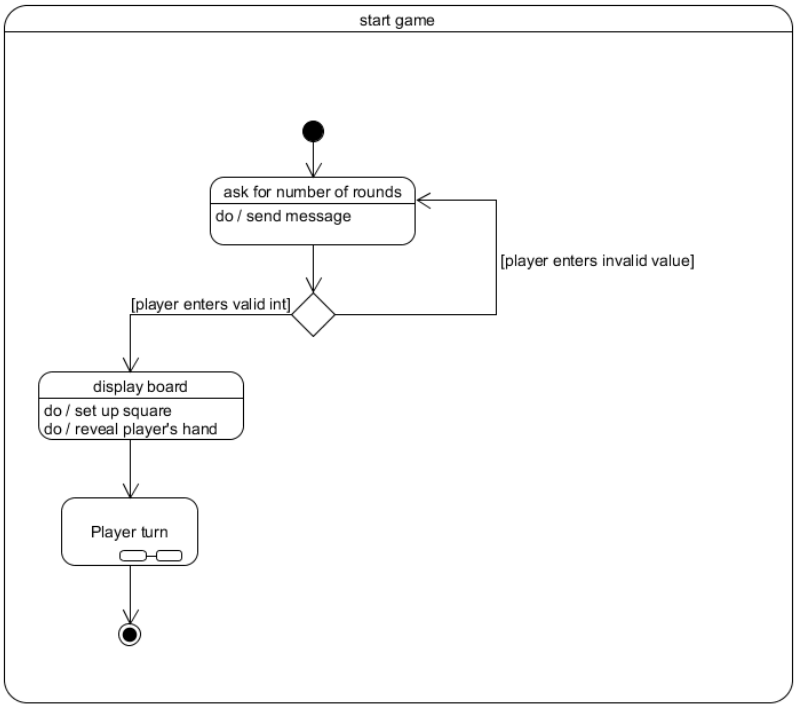


*Figure 23: State diagram - start game*

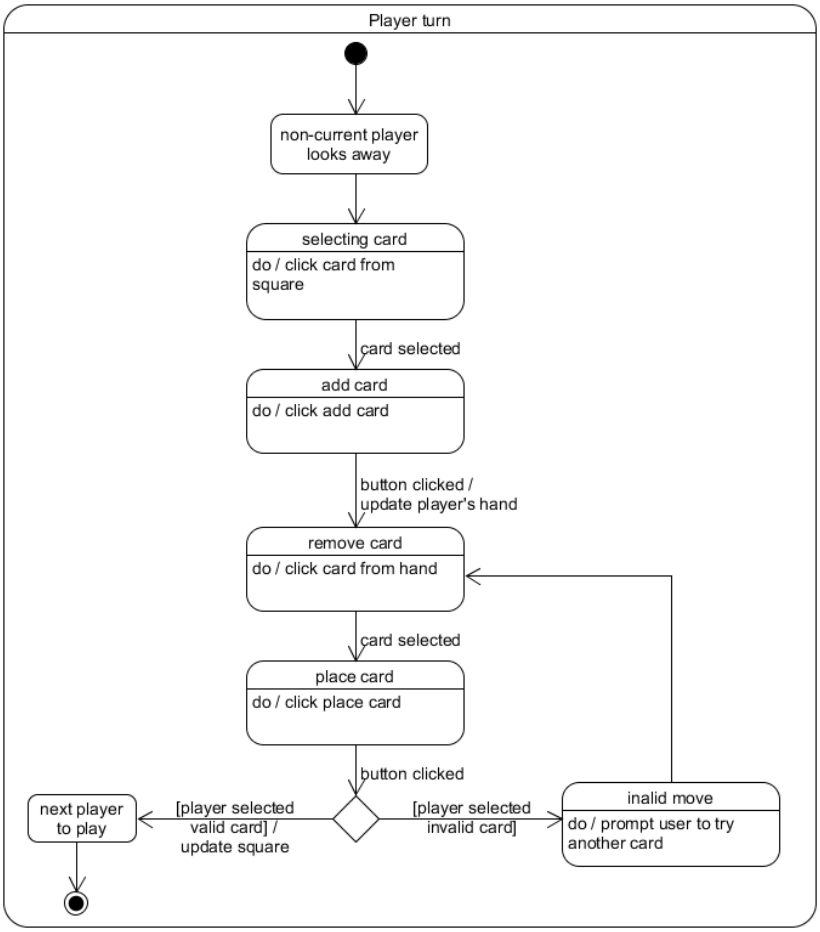


*Figure 24: State Diagram - Player turn*

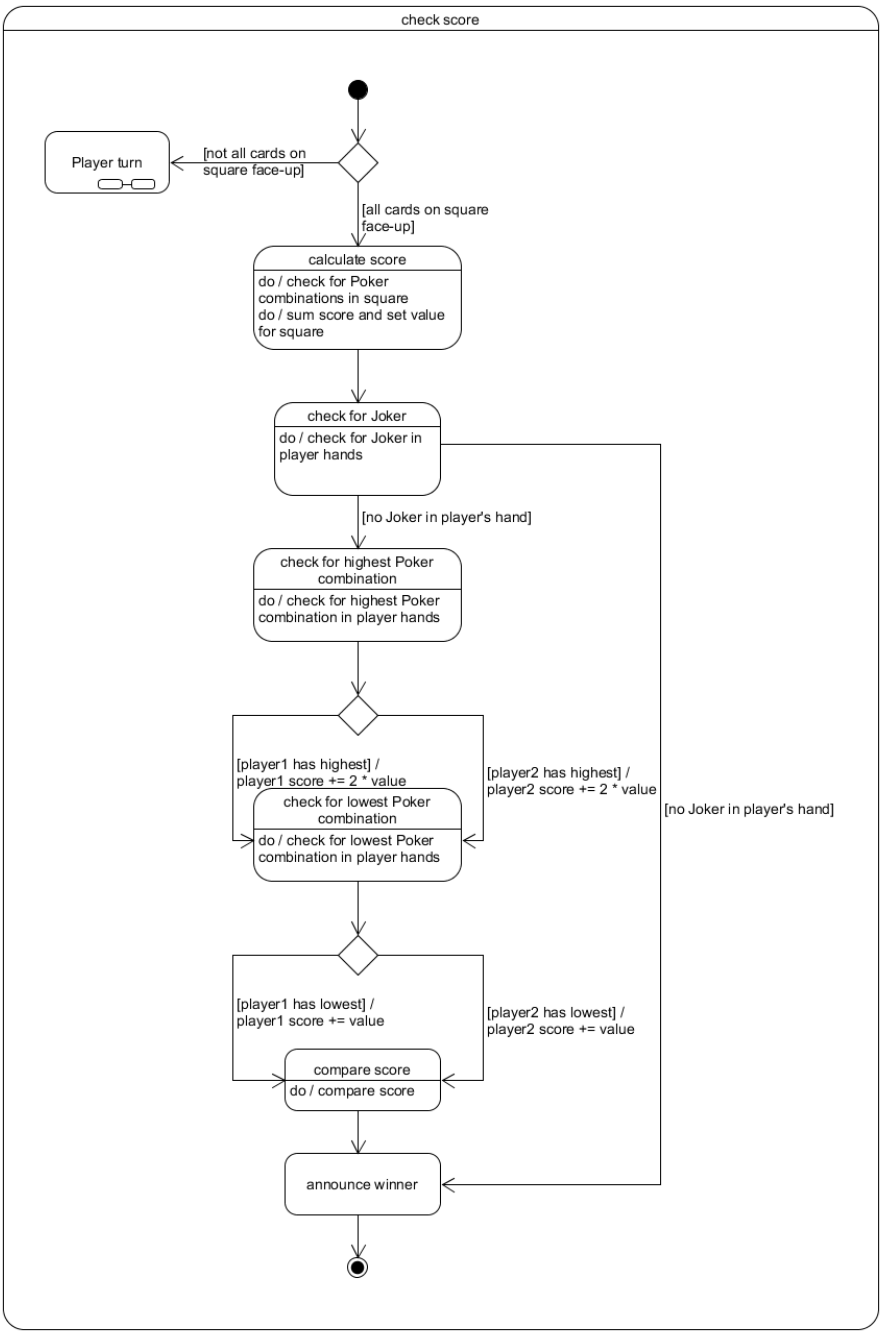


*Figure 25: State diagram - check score*

# Chapter 5: Testing

## 5.1 Acceptance Tests

### 5.1.1 UC01 Tests

UC01 T1: Test that, a player enters a valid integer for number of rounds to be played.

UC01 T2: Test that, the current round of play is set to 1 when the players start playing.

### 5.1.2 UC02 Tests

UC02 T1: Test that, if a player clicks a card is FACE_UP on the square, and the player has no Joker in hand, the system prompts the user to select again. Invalid move.

UC02 T2: Test that, if a player clicks a card is FACE_UP on the square, and the player has Joker in hand, the system prompts the user to select the Joker.

UC02 T3: Test that, if a player clicks the Joker on the square, the replacement card is placed FACE_DOWN.

UC02 T4: Test that, if a player clicks on a normal card on the square, the replacement card is placed FACE_UP.

UC02 T5: Test that, if a player has completed a turn, the current player switches to the other player.

### 5.1.3. UC03 Tests

UC03 T1: Test that, if a player has completed a turn and the cards on the square are not all FACE_UP, that control is transferred to the next player and the game continues.

UC03 T2: Test that, if current round is less than number of rounds decided by the player, the system refreshes for another round

UC03 T3: Test that, if a player holds a Joker in their hand, they lose.

UC03 T4: Test that, if both players hold the same score, the system displays a draw.

UC03 T5: Test that, if a player has no Poker combinations in their hand at the end of the round, they score 0.

## 5.2 Test Cases

### 5.2.1 Test Case: Picking up a card

| Class **Player** | |
|---|---|
| Method<br>`pickUpCard(Card c)` | Receiver and message<br>`player1.pickUpCard(A-HEARTS)` |
| Fixture and input<br><br>A Square object square1 linked to twenty-five card objects (TEN-CLUBS, FIVE-HEARTS, TEN-HEARTS, A-SPADES, SIX-HEARTS, EIGHT-DIAMONDS, FIVE-SPADES, J-HEARTS, FIVE-CLUBS, SEVEN-DIAMONDS, TEN-DIAMONDS, A-DIAMONDS, A-HEARTS, EIGHT-HEARTS, Q-CLUBS, Q-HEARTS, K-SPADES, J-DIAMONDS, SIX-DIAMONDS, NINE-SPADES, NINE-CLUBS, EIGHT-CLUBS, K-DIAMONDS, NINE-HEARTS, JOKER). | Expected result<br><br>square1 is still linked with twenty-five Card objects. |

| | |
|---|---|
| A Player object, player1, is linked to square1.<br><br>All the Card objects linked to square1 have the state as FACE_DOWN. | |

| Class **Player** | |
|---|---|
| Method<br>`pickUpCard(Card c)` | Receiver and message<br>`player1.pickUpCard(JOKER)` |
| Fixture and input<br><br>A Square object square1 linked to twenty-five card objects (TEN-CLUBS, FIVE-HEARTS, TEN-HEARTS, A-SPADES, SIX-HEARTS, EIGHT-DIAMONDS, FIVE-SPADES, J-HEARTS, FIVE-CLUBS, SEVEN-DIAMONDS, TEN-DIAMONDS, A-DIAMONDS, A-HEARTS, EIGHT-HEARTS, Q-CLUBS, Q-HEARTS, K-SPADES, J-DIAMONDS, SIX-DIAMONDS, NINE-SPADES, NINE-CLUBS, EIGHT-CLUBS, K-DIAMONDS, NINE-HEARTS, JOKER).<br><br>A Player object, player1, is linked to square1.<br><br>All the Card objects linked to square1 have the state as FACE_DOWN. | Expected result<br><br>The Card object JOKER has the state FACE_UP.<br><br>square1 is still linked with twenty-five Card objects. |

### 5.2.2 Test Case: Adding a card

| Class **Hand** | |
|---|---|
| Method<br>`addCard(Card c)` | Receiver and message<br>`hand1.addCard(A-HEARTS)` |
| Fixture and input<br><br>A Square object, square1, linked to twenty-five card objects (TEN-CLUBS, FIVE-HEARTS, TEN-HEARTS, A-SPADES, SIX-HEARTS, EIGHT-DIAMONDS, FIVE-SPADES, J-HEARTS, FIVE-CLUBS, SEVEN-DIAMONDS, TEN-DIAMONDS, A-DIAMONDS, A-HEARTS, EIGHT-HEARTS, Q-CLUBS, Q-HEARTS, K-SPADES, J-DIAMONDS, SIX-DIAMONDS, NINE-SPADES, NINE-CLUBS, EIGHT-CLUBS, K-DIAMONDS, NINE-HEARTS, JOKER).<br><br>A Hand object, hand1, is linked to eight Card objects (TEN-SPADES, SEVEN-HEARTS, SIX-SPADES, SEVEN-CLUBS, J-SPADES, FIVE-DIAMONDS, Q-SPADES, K-HEARTS).<br><br>A Player object, player1, is linked to hand1.<br><br>player1 has picked up the Card object, A-HEARTS (player1.pickUpCard(A-HEARTS)).<br><br>The Card object A-HEARTS has the state FACE_UP. | Expected result<br><br>hand1 is linked to nine Card objects and contains the Card object, A-HEARTS.<br><br>square1 is linked to twenty-four Card objects.<br><br>The Card object, A-HEARTS, has the state IN_HAND. |

### 5.2.3 Test Case: Removing a card

| Class **Hand** | |
|---|---|
| Method<br>`removeCard(Card c)` | Receiver and message<br>`hand1.removeCard(SIX-SPADES)` |
| Fixture and input<br><br>A Square object, square1, linked to twenty-four card objects (TEN-CLUBS, FIVE-HEARTS, TEN-HEARTS, A-SPADES, SIX-HEARTS, EIGHT-DIAMONDS, FIVE-SPADES, J-HEARTS, FIVE-CLUBS, SEVEN-DIAMONDS, TEN-DIAMONDS, A-DIAMONDS, null, EIGHT-HEARTS, Q-CLUBS, Q-HEARTS, K-SPADES, J-DIAMONDS, SIX-DIAMONDS, NINE-SPADES, NINE-CLUBS, EIGHT-CLUBS, K-DIAMONDS, NINE-HEARTS, JOKER).<br><br>A Hand object, hand1, is linked to nine Card objects (TEN-SPADES, SEVEN-HEARTS, SIX-SPADES, SEVEN-CLUBS, J-SPADES, FIVE-DIAMONDS, Q-SPADES, K-HEARTS, A-HEARTS).<br><br>A Player object, player1, is linked to hand1. | Expected result<br><br>hand1 is still linked to nine Card objects<br><br>The Card object, SIX-SPADES, has the state FACE_UP |

### 5.2.4 Test Case: Placing a card

| Class **Player** | |
|---|---|
| Method<br>`placeCard(Card c)` | Receiver and message<br>`player1.placeCard(SIX-SPADES)` |
| Fixture and input<br><br>A Square object, square1, linked to twenty-four card objects (TEN-CLUBS, FIVE-HEARTS, TEN-HEARTS, A-SPADES, SIX-HEARTS, EIGHT-DIAMONDS, FIVE-SPADES, J-HEARTS, FIVE-CLUBS, SEVEN-DIAMONDS, TEN-DIAMONDS, A-DIAMONDS, null, EIGHT-HEARTS, Q-CLUBS, Q-HEARTS, K-SPADES, J-DIAMONDS, SIX-DIAMONDS, NINE-SPADES, NINE-CLUBS, EIGHT-CLUBS, K-DIAMONDS, NINE-HEARTS, JOKER).<br><br>A Hand object, hand1, is linked to nine Card objects (TEN-SPADES, SEVEN-HEARTS, SIX-SPADES, SEVEN-CLUBS, J-SPADES, FIVE-DIAMONDS, Q-SPADES, K-HEARTS, A-HEARTS).<br><br>A Player object, player1, is linked to hand1.<br><br>hand1 has removed the Card object, SIX-SPADES (hand1.removeCard(SIX-HEARTS)).<br><br>The Card object SIX-SPADES has the state FACE_UP. | Expected result<br><br>hand1 is linked to eight Card objects and doesn't contain the Card object, SIX-SPADES.<br><br>The Card object, SIX-SPADES, has the state FACE-UP.<br><br>square1 is linked to twenty-five Card objects and contains the Card object, SIX-SPADES. |

### 5.2.5 Test Case: Is the card valid?

| Class **Player** | |
|---|---|
| Method<br>`isValid(Card c1, Card c2)` | Receiver and message<br>`player1.isValid(A-HEARTS, SIX-SPADES)` |
| Fixture and input<br><br>A Square object, square1, linked to twenty-four card objects (TEN-CLUBS, FIVE-HEARTS, TEN-HEARTS, A-SPADES, SIX-HEARTS, EIGHT-DIAMONDS, FIVE-SPADES, J-HEARTS, FIVE-CLUBS, SEVEN-DIAMONDS, TEN-DIAMONDS, A-DIAMONDS, null, EIGHT-HEARTS, Q-CLUBS, Q-HEARTS, K-SPADES, J-DIAMONDS, SIX-DIAMONDS, NINE-SPADES, NINE-CLUBS, EIGHT-CLUBS, K-DIAMONDS, NINE-HEARTS, JOKER).<br><br>A Hand object, hand1, is linked to nine Card objects (TEN-SPADES, SEVEN-HEARTS, SIX-SPADES, SEVEN-CLUBS, J-SPADES, FIVE-DIAMONDS, Q-SPADES, K-HEARTS, A-HEARTS).<br><br>A Player object, player1, is linked to hand1.<br><br>hand1 has removed the Card object, SIX-SPADES (hand1.removeCard(SIX-HEARTS)).<br><br>The Card object A-HEARTS has the state IN_HAND.<br><br>The Card object SIX-SPADES has the state FACE_UP. | Expected result<br><br>Returns true. |

| Class **Player** | |
|---|---|
| Method<br>`isValid(Card c1, Card c2)` | Receiver and message<br>`player1.isValid(JOKER, SIX-SPADES)` |
| Fixture and input<br><br>A Square object, square1, linked to twenty-four card objects (TEN-CLUBS, FIVE-HEARTS, TEN-HEARTS, A-SPADES, SIX-HEARTS, EIGHT-DIAMONDS, FIVE-SPADES, J-HEARTS, FIVE-CLUBS, SEVEN-DIAMONDS, TEN-DIAMONDS, A-DIAMONDS, A-HEARTS, EIGHT-HEARTS, Q-CLUBS, Q-HEARTS, K-SPADES, J-DIAMONDS, SIX-DIAMONDS, NINE-SPADES, NINE-CLUBS, EIGHT-CLUBS, K-DIAMONDS, NINE-HEARTS, null).<br><br>A Hand object, hand1, is linked to nine Card objects (TEN-SPADES, SEVEN-HEARTS, SIX-SPADES, SEVEN-CLUBS, J-SPADES, FIVE-DIAMONDS, Q-SPADES, K-HEARTS, A-HEARTS).<br><br>A Player object, player1, is linked to hand1. | Expected result<br><br>Returns true. |

| | |
|---|---|
| hand1 has removed the Card object, SIX-SPADES (hand1.removeCard(SIX-HEARTS)).<br><br>The Card object JOKER has the state IN_HAND.<br><br>The Card object SIX-SPADES has the state FACE_UP. | |

| Class **Player** | |
|---|---|
| Method<br>`isValid(Card c1, Card c2)` | Receiver and message<br>`player1.isValid(SIX-SPADES, JOKER)` |
| Fixture and input<br><br>A Square object, square1, linked to twenty-four card objects (TEN-CLUBS, FIVE-HEARTS, TEN-HEARTS, A-SPADES, SIX-HEARTS, EIGHT-DIAMONDS, FIVE-SPADES, J-HEARTS, FIVE-CLUBS, SEVEN-DIAMONDS, TEN-DIAMONDS, A-DIAMONDS, A-HEARTS, EIGHT-HEARTS, Q-CLUBS, Q-HEARTS, K-SPADES, J-DIAMONDS, SIX-DIAMONDS, NINE-SPADES, NINE-CLUBS, EIGHT-CLUBS, K-DIAMONDS, NINE-HEARTS, null).<br><br>A Hand object, hand1, is linked to nine Card objects (TEN-SPADES, SEVEN-HEARTS, SIX-SPADES, SEVEN-CLUBS, J-SPADES, FIVE-DIAMONDS, Q-SPADES, K-HEARTS, A-HEARTS).<br><br>A Player object, player1, is linked to hand1.<br><br>hand1 has removed the Card object, SIX-SPADES (hand1.removeCard(JOKER)).<br><br>The Card object JOKER has the state FACE_UP.<br><br>The Card object SIX-SPADES has the state IN_HAND. | Expected result<br><br>Returns true. |

| Class **Player** | |
|---|---|
| Method<br>`isValid(Card c1, Card c2)` | Receiver and message<br>`player1.isValid(SIX-SPADES, JOKER)` |
| Fixture and input<br><br>A Square object, square1, linked to twenty-four card objects (TEN-CLUBS, FIVE-HEARTS, TEN-HEARTS, A-SPADES, SIX-HEARTS, EIGHT-DIAMONDS, FIVE-SPADES, J-HEARTS, FIVE-CLUBS, SEVEN-DIAMONDS, TEN-DIAMONDS, A-DIAMONDS, A-HEARTS, EIGHT-HEARTS, Q-CLUBS, Q-HEARTS, K-SPADES, J-DIAMONDS, SIX-DIAMONDS, NINE-SPADES, NINE-CLUBS, EIGHT-CLUBS, K-DIAMONDS, NINE-HEARTS, null). | Expected result<br><br>Returns false. |

| | |
|---|---|
| A Hand object, hand1, is linked to nine Card objects (TEN-SPADES, SEVEN-HEARTS, SIX-SPADES, SEVEN-CLUBS, J-SPADES, FIVE-DIAMONDS, Q-SPADES, K-HEARTS, A-HEARTS).<br><br>A Player object, player1, is linked to hand1.<br><br>hand1 has removed the Card object, SIX-SPADES (hand1.removeCard(JOKER)).<br><br>The Card object JOKER has the state FACE_DOWN.<br><br>The Card object SIX-SPADES has the state IN_HAND. | |

### 5.2.6 Test Case: Does the player's hand contain a Joker?

| Class **Player** | |
|---|---|
| Method<br>`jokerInHand(Hand h)` | Receiver and message<br>`player1.jokerInHand(hand1)` |
| Fixture and input<br><br>A Hand object, hand1, is linked to eight Card objects (TEN-SPADES, SEVEN-HEARTS, SEVEN-CLUBS, J-SPADES, FIVE-DIAMONDS, Q-SPADES, K-HEARTS, A-HEARTS).<br><br>A Player object, player1, is linked to hand1. | Expected result<br><br>Returns false. |

| Class **Player** | |
|---|---|
| Method<br>`jokerInHand(Hand h)` | Receiver and message<br>`player1.jokerInHand(hand1)` |
| Fixture and input<br><br>A Hand object, hand1, is linked to eight Card objects (TEN-SPADES, SEVEN-HEARTS, SEVEN-CLUBS, JOKER, FIVE-DIAMONDS, Q-SPADES, K-HEARTS, A-HEARTS).<br><br>A Player object, player1, is linked to hand1. | Expected result<br><br>Returns true. |

### 5.2.7 Test Case: Getting the highest Poker combination from a hand

| Class **Hand** | |
|---|---|
| Method<br>`makeHighCombo()` | Receiver and message<br>`hand1.makeHighCombo()` |
| Fixture and input<br><br>A Hand object, hand1, is linked to eight Card objects (7-HEARTS, 8-HEARTS, 9-HEARTS, 10-HEARTS, J-HEARTS, Q-HEARTS, K-HEARTS, A-HEARTS). | Expected result<br><br>Returns `new PokerCombo(Combination.STRAIGHT_FLUSH, 10)` |

### 5.2.8 Test Case: Getting the lowest Poker combination from a hand

| Class **Hand** | |
|---|---|
| Method<br>`makeLowCombo()` | Receiver and message<br>`hand1.makeLowCombo()` |
| Fixture and input<br><br>A Hand object, hand1, is linked to eight Card objects (7-HEARTS, 8-HEARTS, 9-HEARTS, 10-HEARTS, J-HEARTS, Q-HEARTS, K-HEARTS, A-HEARTS). | Expected result<br><br>Returns `new PokerCombo(Combination.FLUSH, 5)` |

### 5.2.9 Test Case: Does a hand contain a Joker?

| Class **Hand** | |
|---|---|
| Method<br>`containsJoker()` | Receiver and message<br>`hand1.containsJoker()` |
| Fixture and input<br><br>A Hand object, hand1, is linked to eight Card objects (7-HEARTS, 8-HEARTS, 9-HEARTS, 10-HEARTS, J-HEARTS, Q-HEARTS, K-HEARTS, A-HEARTS). | Expected result<br><br>Returns false. |

### 5.2.10 Test Case: Compare score between two players

| Class **Game** | |
|---|---|
| Method<br>`compareScore(Player p1, Player p2)` | Receiver and message<br>`game.compareScore(Player player1, Player2 player2)` |
| Fixture and input<br><br>A Game object , game, is linked to List<Player> players, which contains references to two Player objects, player1 and player2.<br><br>player1 has a score of 432.<br><br>player2 has a score of 328. | Expected result<br><br>Returns `player1` |