System design report

By Benjamin Adjei

Contents

[Chapter 1: Requirements 2](#_Toc511042716)

[Domain description 2](#_Toc511042717)

[Functional requirements 2](#_Toc511042718)

[Quality requirements 2](#_Toc511042719)

[Use case: Begin the card game Ganderpoke 3](#_Toc511042720)

[Use case: See if a player’s won the game 3](#_Toc511042721)

[Use case: Take a turn 3](#_Toc511042722)

[Workflow illustrations 4](#_Toc511042723)

[Chapter 2: Initial conceptual model 5](#_Toc511042724)

[Chapter 3: Dynamic design 7](#_Toc511042725)

[Chapter 4: User interface interaction 11](#_Toc511042726)

[Chapter 5: Testing 12](#_Toc511042727)

# Chapter 1: Requirements for the Ganderpoke card game

## 1.1 Context

The objective of Ganderpoke is for a player to make either the lowest or highest Poker combination with the cards in their hand. The total score of a player’s hand is the sum of the scores of the rows and columns at the end of the game. Below is a description of the gameplay

## 1.2 Domain description (instructions for playing the game)

The player begins with a pack of 52 cards, then they add a Joker and take all 2’s, 3’s and 4’s. The pack now has 41 cards. 25 cards are dealt face down on the table in a 5x5 square. The last 16 cards are shared face down between the 2 players.

The player who isn’t dealing the cards starts, and each player takes turns to play. On a turn, each player can get a card that’s face down in the square and replace it with a face up card from their hand, which might be the card just taken.

If a Joker is in a player’s hand, on their turn they can replace a card that faces up from the square with their Joker with the face up which ends the player’s go. The next player then has a turn.

If a Joker is face up in the square, on their go, the player might replace the Joker with a face down card from their hand.

If a Joker is in a player’s hand at the end of play, that player is the loser.

The game ends when all 25 cards in the square are facing up. Each player’s score is calculated in the following way: every column and row are thought of as a 5-card Poker combination. A pair of cards is worth a single point, 2 pairs are 2 points, a triplet is 3 points, a straight, flush, or full house is 5 points, 4 of a kind is 8 points and a straight flush is 10 points. Jokers in the square are worth the same amount of points of the card needed to get the highest possible combination in both directions. The total score for the square is the sum of the scores for the ten Poker combinations.

The player who makes the highest-scoring 5-card Poker hand scores 2 times the square’s value and the player who makes the lowest scoring 5-card combination, this may be the same person, scores the square’s value. This makes up a round of play. Players choose how many rounds of play there are going to be when the game starts

The player who scores the highest is the winner of the game.

## 1.3 Requirements overview

## 1.3.1 Product overview

Theoretically, the objective is to make a computer game version of the Ganderpoke game described above. The idea is the game is played by 2 players side-by-side at a PC or laptop taking turns with a keyboard or mouse.

## 1.3.2 Stakeholders

The tutor and the module students are the stakeholders in this theoretical scenario. In a real project, there would be many stakeholders which would include at least the client and software development company.

## 1.3.3 Product functionality

The functionality must be described at nearly a high level because it’s required at the early stage of requirements analysis with a view to start modelling conceptually.

## 1.3.4 Scope

The system will aid learning and teaching with respect to the 2nd block of the module.

## 1.4 Workflow models

These represent how the game works, starting with the high-level model in Figure 1.1 below. There a sub-activity models to make the high-level workflow less cluttered. There’s no single solution so these can vary.

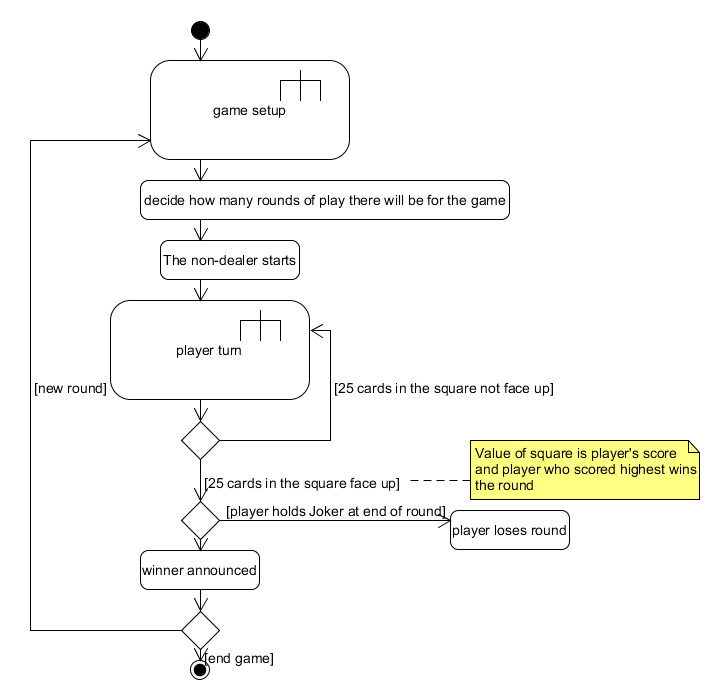


Figure High level workflow of the Ganderpoke card game

This workflow shows how to set up a game where players take turns until a win condition is met and it gives the option of starting a new game after the last one is done. The next workflow example (see Figure 2) shows what happens when a player takes a turn.

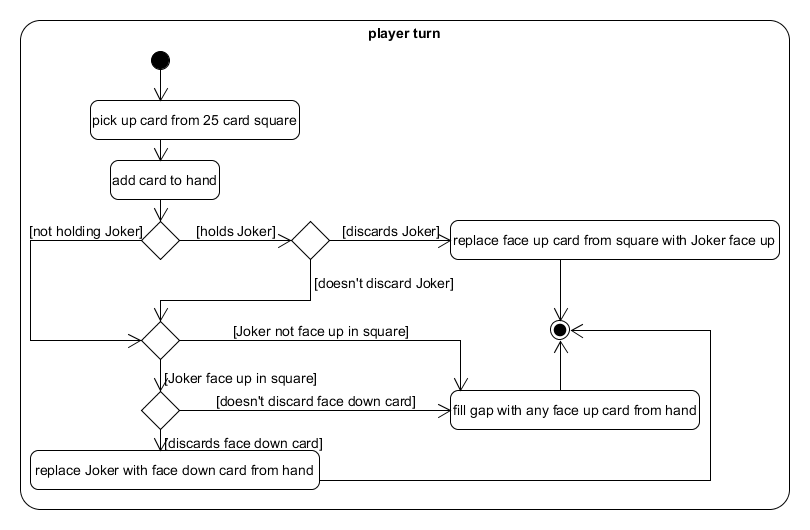


Figure Player turn workflow

The next workflow is a more detailed diagram of the game setup.

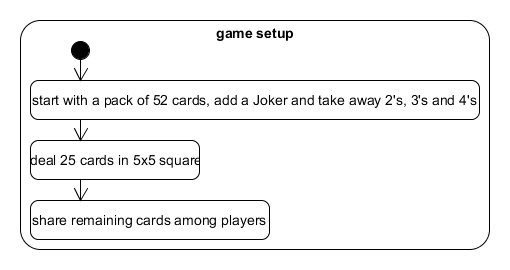


Figure Game setup workflow

## 1.5 Requirements description

Requirements are written in different ways e.g. Volere templates, user stories or use cases. In this section, the requirements are written as unambiguous single sentences. There can be variations on how to do this.

## 1.5.1 Functional requirements

From the game server perspective:

* Start a game of Ganderpoke with 2 players
* Register a player with a game
* Commence a game
* Add Joker and remove 2’s, 3’s and 4’s from pack
* Deal cards face down in a 5x5 square
* Share remaining cards among players
* Record a player move (e.g. a player picks up a card)
* Check whether moves are legal according with the rules of the game
* Feedback the state of the current game to the players, i.e. signalling whose turn it is
* Announce the winner
* Reset to play another game

From the perspective of a player

* Connect to a game service
* Play the game
* Pick up a card from the square
  + Discard a card from your hand
* Exit the game

These functional requirements would have come from a discussion with the client.

## 

## 1.5.2 Quality requirements (small sample)

These have been included for reference.

* The application will be implemented using Java Standard Edition, Version 8.
* The application will run on Windows

## 1.7 Use cases

## 

## 1.7.1 Use case: Begin the card game Ganderpoke

Name, identifier, version: Begin the card game Ganderpoke, UC01, Version 1.0

Initiator: game administrator

Goal: begin a game

Assumptions: two players are side-by-side at a PC or laptop with a keyboard

Main scenario:

1. The players begin with a pack of 52 cards
2. A Joker is added to the game pack and 2’s, 3’s and 4’s are taken from the game pack
3. 25 cards are dealt face down in a 5x5 square and the last 16 cards are shared face down among the 2 players
4. The person who didn’t deal the cards starts

## 1.7.2 Use case: See if a player’s won the game

Name, identifier, version: See who’s won the game, UC02, Version 1.0

Initiator: game administrator

Goal: to check the game to see who’s won

Assumptions: the game is over

Main scenario:

1. If a Joker is in a player’s hand when play finishes, the player who’s holding the Joker loses
2. A check is made to see which player has scored the highest
3. If one player’s score is higher than the other player, the player whose score is highest is declared the winner

## 1.7.3 Use case: Take a turn

Name, identifier, version: Take a turn, UC03, Version 1.0

Initiator: player

Goal: to discard 1 card that the player has

Assumptions: it’s the turn of the current player and no player has yet won the game

Main scenario:

1. The player replaces a card that’s face-down in the square with a card in their hand that’s face up

Extensions

1a If a Joker is in a player’s hand, on their turn, they might replace a face up card in the square with the Joker face up. This finishes the player’s turn.

1b If a Joker is face up in the square, on their turn, the player might replace it with a face down card from their hand

## 1.8 Acceptance tests (sample only)

Instead of theoretically, the requirements phase would consist of the stakeholder/client making the use cases for the system and doing many ‘acceptance tests’. These validate the system when the developer makes a version for testing available. The tests ensure the product works the way the client wants it to work.

## 1.8.1 UC02 example acceptance tests

UC02 T1: Test that, if one player’s score is higher than another at the end of the game and that player doesn’t hold the Joker, that player is declared the winner

UC02 T2: Test that, if one player’s turn is over and their hand has 8 cards, that the next player takes a turn and the game continues

These tests assure the client that system operates in the way that the client wants.

Following are more examples

## 1.8.2 UC03 example acceptance tests

UC03 T1: Test that, if a player holds more than one card in their hand and doesn’t hold the Joker, a face down card is taken from the square, added to the player’s hand and a card from the player’s hand is discarded face up. Then the player’s turn ends

UC03 T2: Test that, if a player holds the Joker, the player’s turn ends

UC)3 T3: Test that, if the Joker’s face up in the square, the player replaces it with a face down card from their hand

# Chapter 2: Initial conceptual model

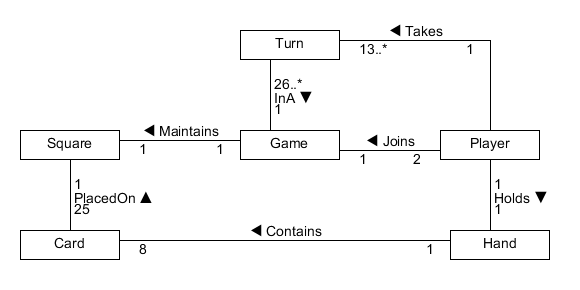


Figure Class diagram of the classes in my Ganderpoke program

Class descriptions

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

**Attributes**

currentSuit The suit of the cards in a player’s

hand (Hearts, Clubs, Diamonds or Spades)

currentRank The rank of the cards in a

player’s hand (Ace to King)

type A card’s type (Normal card or

Joker)

**Class** Player A person who’s playing the game

**Attributes**

hand A player’s hand of cards

**Class** Hand A group of cards a player’s

holding in a game

**Attributes**

numCards How many cards in a player’s

hand

**Class** Square The area the game is played in

**Attributes**

value The value of the square

**Class** Turn Players take this in the game

**Attributes**

None

**Class** Game The game of Ganderpoke

**Attributes**

suit The suits of the cards in the

square

rank The rank of the card in the

square

Invariants

1. If a Player object aPlayer is linked to a Turn object aTurn, and the Turn object aTurn is linked to a Game object aGame, then aPlayer must be linked to aGame.

# Chapter 3: Dynamic design

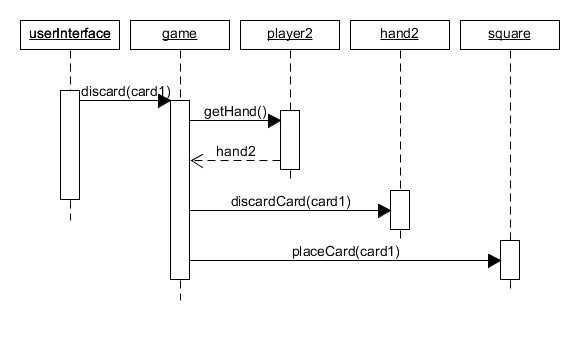


Figure A sequence diagram for discarding a card in the Ganderpoke game

Identifier OS 01, version 1.0

Context: Player

Signature: discard(Card : aCard) : void

Invariant: true

Precondition:

It’s the current player’s turn and they haven’t filled the gap with a card in their

hand

Postcondition:

The gap in the square is filled

The player’s hand contains 1 less card

It’s the next player turn

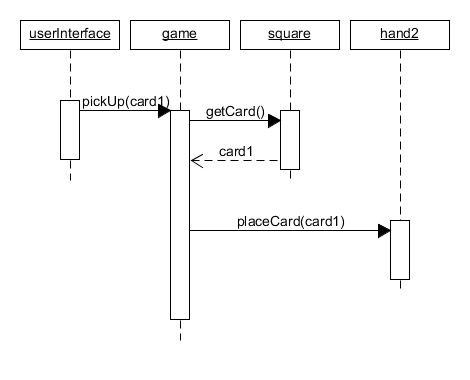


Figure A sequence diagram for picking up a card in the Ganderpoke game

Identifier OS 02, version 1.0

Context: Game

Signature: pickUp() : void

Invariant: true

Precondition:

It’s the current player’s turn and they haven’t picked up a card

Postcondition:

The player’s hand contains 1 more card

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

**Attributes**

currentSuit The suit of the cards in a player’s

hand (Hearts, Clubs, Diamonds or Spades)

currentRank The rank of the cards in a

player’s hand (Ace to King)

**Protocol**

Suit getSuit()

Rank getRank()

CardType getType()

**Class** Player A person who’s playing the game

**Attributes**

None

**Links**

Hand playersHand

The linked Hand object is referenced

Game game

The linked Game object is referenced

**Protocol**

Hand getHand()

void discard (Card card)

void pickUp()

**Class** Hand A group of cards a player’s

holding in a game

**Attributes**

numCards How many cards in a player’s

hand

**Links**

Collection<Card> cards

A group of linked Card objects is referenced

**Protocol**

void removeCard (Card card)

void addCard (Card card)

void setHand(List <Card>)

void discard(Card card)

**Class** Square The area the game is played in

**Attributes**

value The value of the square

**Links**

Collection<Card> cards

A group of linked Card objects

**Protocol**

void placeCard (Card card)

int calculateCard (Card card)

void uncoverCard (Card card)

void removeCard (Card card)

void replaceCard (Card card)

**Class** Turn Players take this in the game

**Attributes**

None

**Class** Game The game of Ganderpoke

**Attributes**

suit The suits of the cards in

the square

rank The ranks of the cards in

the square

**Links**

Collection<Player> players

A collection of linked Player objects is referenced

**Protocol**

void discard (Card card)

void pickup()

Player getCurrentPlayer()

int getRound()

int getHighestScore(Square s)

int getLowestScore(Square s)

int getSquareValue(Square s)

The class descriptions have evolved during dynamic design because there are link references to other classes and protocols/methods for the classes which were found during dynamic design.

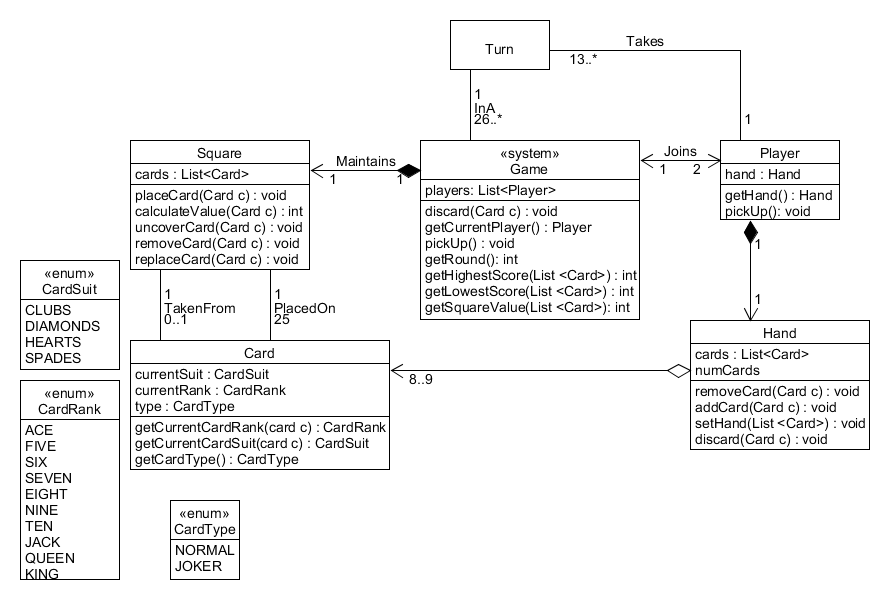


Figure An implementation model of my Ganderpoke program

# Chapter 4: User interface interaction

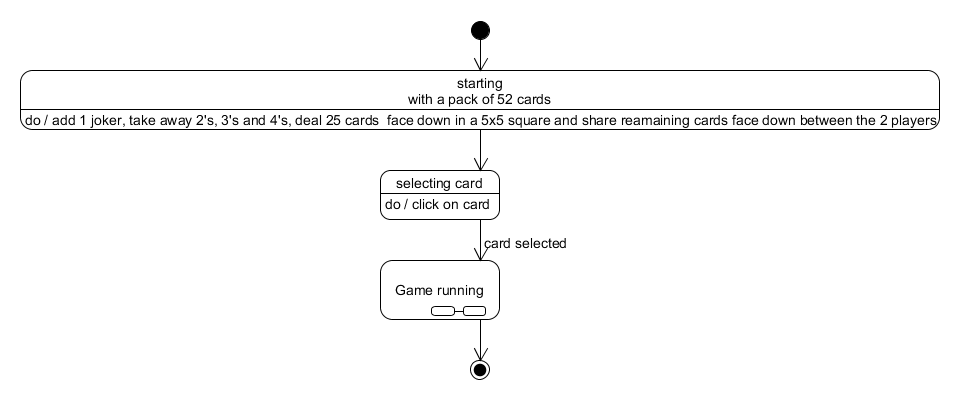


Figure Ganderpoke: start game states

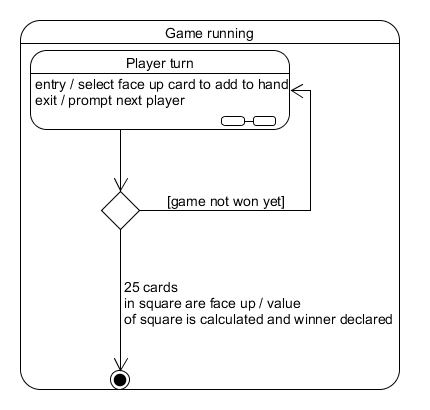


Figure Ganderpoke: mid-game state

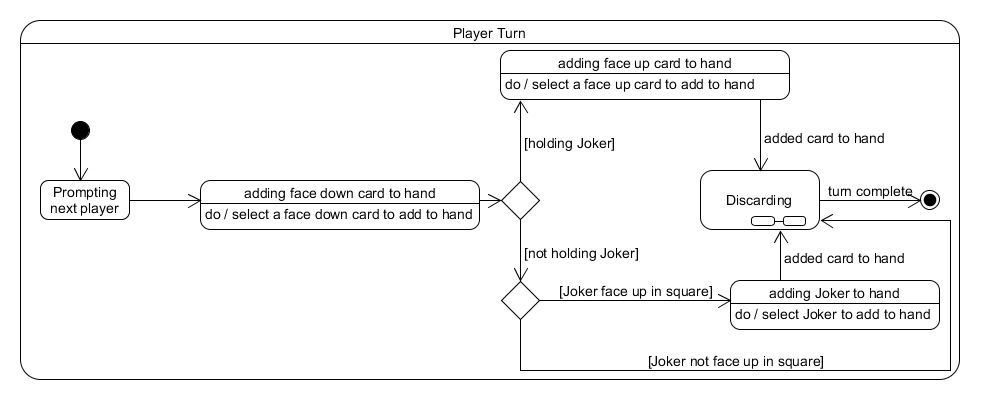


Figure Ganderpoke: turn states

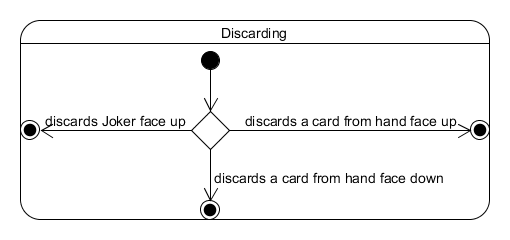


Figure Ganderpoke: discard states

# Chapter 5: Testing

I will test if the 25 cards are random and if the score is calculated correctly at the end of the game.