Christy Gilliland

Patrick Smith

Alexander Burch

Erik Freburger

CMSC 495

10/10/2016

Christy Gilliland

Patrick Smith

Alexander Burch

Erik Freburger

CMSC 495

10/10/2016

Christy Gilliland

Patrick Smith

Alexander Burch

Erik Freburger

CMSC 495

10/10/2016

Christy Gilliland

Patrick Smith

Alexander Burch

Erik Freburger

CMSC 495

10/10/2016

P.A.C.E Card Game Suite

Final Report

**Table of Contents**

[List of Abbreviations and Acronyms 8](#_Toc463902509)

[1. Overview 9](#_Toc463902510)

[1.1. Project Requirements 9](#_Toc463902511)

[1.2. Project Summary 9](#_Toc463902512)

[1.3. Individual Contributions 10](#_Toc463902513)

[2. Project PLAN 11](#_Toc463902514)

[2.1. Project Summary 11](#_Toc463902515)

[2.1.1. Scope and Objective 11](#_Toc463902516)

[2.1.2. Assumptions and Constraints 11](#_Toc463902517)

[2.1.3. Project Deliverables 12](#_Toc463902518)

[2.2. References (Applicable to Project Plan) 12](#_Toc463902519)

[2.3. Project Organization 13](#_Toc463902520)

[2.3.1. External Interfaces 13](#_Toc463902521)

[2.3.2. Internal Structure 13](#_Toc463902522)

[2.3.1. Roles and Responsibilities 14](#_Toc463902523)

[2.4. Managerial Process Plan 15](#_Toc463902524)

[2.4.1. Work Plan 15](#_Toc463902525)

[2.4.2. Risk Management Plan 16](#_Toc463902526)

[2.5. Technical Process Plan 17](#_Toc463902527)

[2.5.1.Process Model 17](#_Toc463902528)

[2.5.2. Methods, Tools, and Techniques 18](#_Toc463902529)

[3. System specification 19](#_Toc463902530)

[3.1. External Interface Requirements 19](#_Toc463902531)

[3.1.1. User Interfaces 19](#_Toc463902532)

[3.1.2. Hardware Interfaces 19](#_Toc463902533)

[4. requirements specification 20](#_Toc463902534)

[4.1. Performance Requirements 20](#_Toc463902535)

[4.1.1. Capacity 20](#_Toc463902536)

[4.2. Safety Requirements 20](#_Toc463902537)

[4.3. Security Requirements 20](#_Toc463902538)

[4.4. Dynamic Quality Requirements 20](#_Toc463902539)

[4.5. Software Requirements 21](#_Toc463902540)

[4.5.1. Object: Card 21](#_Toc463902541)

[4.5.2. Object: Hand 23](#_Toc463902542)

[4.5.3. Object: Deck 24](#_Toc463902543)

[4.5.4. Object: War 28](#_Toc463902544)

[4.5.5. Object: Concentration 30](#_Toc463902545)

[4.5.6. Object: Blackjack 31](#_Toc463902546)

[4.5.7. Object: HighLow 33](#_Toc463902547)

[4.5.8. Object: ThirtyOne 34](#_Toc463902548)

[4.5.9. Object: WarFrame 40](#_Toc463902549)

[4.5.10. Object: Splash 41](#_Toc463902550)

[4.5.11. Object: SplashScreenPanel 41](#_Toc463902551)

[4.5.12. Object: group3 42](#_Toc463902552)

[5. design 43](#_Toc463902553)

[5.1. Product Perspective 43](#_Toc463902554)

[5.2. Product Functions 43](#_Toc463902555)

[5.2.1. Initial Game Selection Menu 43](#_Toc463902556)

[5.2.2. Game Load Menu 44](#_Toc463902557)

[5.2.3. Game Save Menu 44](#_Toc463902558)

[5.2.4. Individual Game Displays 44](#_Toc463902559)

[5.2.5. War 44](#_Toc463902560)

[5.2.6. Concentration 44](#_Toc463902561)

[5.2.7. Blackjack 45](#_Toc463902562)

[5.2.8. High or Low 46](#_Toc463902563)

[5.2.8. High or Low 46](#_Toc463902564)

[5.2.9. Thirty-One 46](#_Toc463902565)

[5.2.10. Solitaire 46](#_Toc463902566)

[5.3. User Characteristics 46](#_Toc463902567)

[5.3.1. User Scenarios 47](#_Toc463902568)

[5.4. Constraints 47](#_Toc463902569)

[5.5. Assumptions and Dependencies 48](#_Toc463902570)

[6. user’s guide 48](#_Toc463902571)

[7. test plan and results 48](#_Toc463902572)

[7.1. Test Plan Identifier 48](#_Toc463902573)

[7.2. Objectives 48](#_Toc463902574)

[7.3. Scope 49](#_Toc463902575)

[7.4. References (Applicable to the Software Test Plan) 49](#_Toc463902576)

[7.5. Architectural Context Diagram Mappings 49](#_Toc463902577)

[7.5.1. User Interface 49](#_Toc463902578)

[7.6. Test Items 51](#_Toc463902579)

[7.6.1. Test Case Uniformity Standard 51](#_Toc463902580)

[7.7. Environmental Needs 53](#_Toc463902581)

[7.7.1. Hardware 53](#_Toc463902582)

[7.7.2. Software 53](#_Toc463902583)

[7.8. User Test Cases 53](#_Toc463902584)

[7.8.1. Unified GUI 54](#_Toc463902585)

[7.8.2. War! 55](#_Toc463902586)

[7.8.3. Concentration 56](#_Toc463902587)

[7.8.4. Blackjack 57](#_Toc463902588)

[7.8.5. Thirty-One 59](#_Toc463902589)

[7.8.6. High or Low 60](#_Toc463902590)

[7.8.7. Solitaire 61](#_Toc463902591)

[7.9. Test Deliverables 63](#_Toc463902592)

[7.9.2. Testing Tables 63](#_Toc463902593)

[8. development history 63](#_Toc463902594)

[9. conclusions 64](#_Toc463902595)

[9.1. Lessons Learned 64](#_Toc463902596)

[9.2. Design Strengths 64](#_Toc463902597)

[9.3. Limitations 64](#_Toc463902598)

[9.4. Suggestions for Future Improvement 64](#_Toc463902599)

[Appendix a A-1](#_Toc463902600)

[Appendix B B-1](#_Toc463902601)

**Table of Figures**

[Figure 1 - Group 3 Project Team 13](#_Toc463902469)

[Figure 2 - Project Timeline 16](#_Toc463902470)

[Figure 3 - Main User Interface 19](#_Toc463902471)

[Figure 4 - MainCGS UML Class Diagram 21](#_Toc463902472)

[Figure 5 - Example Card Object 22](#_Toc463902473)

[Figure 6 - Data Flow Diagram Level 1 43](#_Toc463902474)

[Figure 7 - ACD of User Interface 50](#_Toc463902475)

[Figure 8 - ACD of user interface with game 50](#_Toc463902476)

**Table of Tables**

[Table 1 - Individual Roles and Contributions 11](#_Toc463902477)

[Table 2 - Work Breakdown Schedule 15](#_Toc463902478)

[Table 3 - Risk Management Matrix 17](#_Toc463902479)

[Table 4 - Card Methods 22](#_Toc463902480)

[Table 5 - Hand Methods 24](#_Toc463902481)

[Table 6 - Deck Methods 25](#_Toc463902482)

[Table 7 - Game Methods 26](#_Toc463902483)

[Table 8 - HighScore Methods 27](#_Toc463902484)

[Table 9 - HighScores Methods 27](#_Toc463902485)

[Table 10 - Bet Methods 28](#_Toc463902486)

[Table 11 - War Methods 29](#_Toc463902487)

[Table 12 - Concentration Methods 31](#_Toc463902488)

[Table 13 - Blackjack Methods 33](#_Toc463902489)

[Table 14 - HighLow Methods 34](#_Toc463902490)

[Table 15 - ThirtyOne Methods 35](#_Toc463902491)

[Table 16 - Solitaire Methods 36](#_Toc463902492)

[Table 17 - Foundation Methods 37](#_Toc463902493)

[Table 18 - SolitairePile Methods 37](#_Toc463902494)

[Table 19 - Tableau Methods 38](#_Toc463902495)

[Table 20 - Waste Methods 39](#_Toc463902496)

[Table 21 - WarFrame Methods 40](#_Toc463902497)

[Table 22 - Splash Methods 41](#_Toc463902498)

[Table 23 - SplashScreenPanel Methods 41](#_Toc463902499)

[Table 24 - MainCGS Methods 42](#_Toc463902500)

[Table 25 - Test Case Template 53](#_Toc463902501)

[Table 26 - Unified GUI Test Table 54](#_Toc463902502)

[Table 27 - War! Test Table 55](#_Toc463902503)

[Table 28 - Concentration Test Table 57](#_Toc463902504)

[Table 29 - Blackjack Test Table 58](#_Toc463902505)

[Table 30 - Thirty-One Test Table 59](#_Toc463902506)

[Table 31 - High or Low Test Table 60](#_Toc463902507)

[Table 32 - Solitaire Test Table 62](#_Toc463902508)

**List of Abbreviations and Acronyms**

ACD – Architectural Context Diagram

CGS – Card Game Suite

DFD – Data Flow Diagram

GUI – Graphical User Interface

SDP – Software Design Plan

SRS – System Requirements Specification

STS – Software Test Specification

1. overview
   1. Project Requirements

The purpose of the assigned project is to collaborate with a group of peers to design, develop, test, and deliver a project that represents the level of skill obtained throughout the course of a Bachelor’s Degree.

* 1. Project Summary

Group 3’s project is herein known as the “P.A.C.E Card Game Suite”. The P.A.C.E Card Game Suite (CGS) is a collection of popular card games designed with graphical interfaces and typical user-friendly features, such as game saves and high scores lists. The P.A.C.E Card Game Suite includes a full version of each of the following games:

* War
* Concentration
* High or Low
* Blackjack
* Thirty-One
* Solitaire.

A central graphical user interface (GUI) allows users to navigate to each game quickly. Each game in the suite includes a custom menu to provide users with the ability to start a new game, save an existing game, load a saved game, view high scores, and display instructions for gameplay. The P.A.C.E Card Game Suite also includes a full color User’s Guide, complete with in-depth “How to Play” instructions and visual aids for each game.

* 1. Individual Contributions

| Lead | Role(s) | Contributions |
| --- | --- | --- |
| Christy Gilliland | Project Lead  Documentation Lead  Coding | * Phase reports * Concentration.java * HighLow.java * Created graphic files to represent cards * Created and maintained the User’s Guide * Organized bi-weekly group chats * Maintained communication with the Professor on behalf of group 3 * Created weekly discussions for each week’s assignments * Final Report |
| Patrick Smith | Coding Lead  Requirements Lead | * Created the original System Requirements Specification (SRS) * Designed most of the major classes in the suite:   + Bet.java   + Blackjack.java   + Card.java   + Deck.java   + Foundation.java   + Game.java   + Hand.java   + HighScore.java   + HighScores.java   + Rank.java   + Solitaire.java   + SolitairePile.java   + Suit.java   + Tableau.java   + WarFrame.java   + Waste.java * Supported Christy and Alex to update other java files to reuse the classes and implement save/load game functionality * Created Splash screen and graphic. |
| Alexander Burch | Design Lead  Coding | * MainCGS.java * War.java * Thirty-One.java * Created main menu graphics * Created Project Design Plan, incorporating Patrick’s SRS * Updated Design Plan to be included in Final Report |
| Erik Freburger | Testing Lead | * Created Software Design Plan (SDP) * Created Software Test Specification (STS) * Designed and Performed Test Cases * Updated STS to be included in Final Report |

Table - Individual Roles and Contributions

1. Project Plan
   1. Project Summary
      1. Scope and Objectives

The objective and scope of the P.A.C.E Card Game Suite is to provide an engaging and entertaining variation of card games for our customers. In order to accommodate the range of games available to customers in a user-friendly manner, a unified GUI will act as the launching point for all game applications.

* + 1. Assumptions and Constraints

It is assumed that end users will know how to use a GUI-based computer application. Additionally, it is assumed the user will know how to download, install and run the CGS; and that the user has a computer system that can handle the resources the CGS requires.

Constraints for the system will be minimal. The CGS cannot run without Java and must have read/write authorization for the folder where the game is located. The CGS and User’s Guide is only available in English.

* + 1. Project Deliverables

The project deliverables for the P.A.C.E Card Game Suite include full versions of:

* War
* Concentration
* High or Low
* Blackjack
* Thirty-One
* Solitaire

It also includes the following documentation:

* Software Design Plan (SDP)
* Software Test Specification (STS)
* User’s Guide
* Software Requirements Specification (SRS) (Project Design).
  1. References (Applicable to Project Plan)
* IEEE. IEEE Std 1058-1998 IEEE Recommended Practice for Software Design Descriptions. IEEE Computer Society, 1998.
* Java EE SDK 6 and JDK 6 Downloads. (n.d.). Retrieved August 31, 2016, from http://www.oracle.com/technetwork/java/javaee/downloads/java-ee-sdk-6u3-jdk-6u29-downloads-523388.html
* Warlordpat/CMSC-495-Current-Trends-and-Projects-in-Computer-Science. (n.d.). Retrieved August 31, 2016, from <https://github.com/warlordpat/CMSC-495-Current-Trends-and-Projects-in-Computer-Science>
  1. Project Organization

The organization of the project is split into two separate entities. The entities are the external and internal personnel needed to complete the project. It is important to note project team members serve in both capacities.

* + 1. External Interfaces

The external interface will consist of one individual, Christy Gilliland, who will act as the liaison between the project team and professor. This individual will act as the single source of communication between the professor and project team. The project lead will communicate clearly and concisely any needs or concerns to the professor and vice versa. Lastly, the project lead communicates to the professor any issues the team is experiencing.

* + 1. Internal Structure

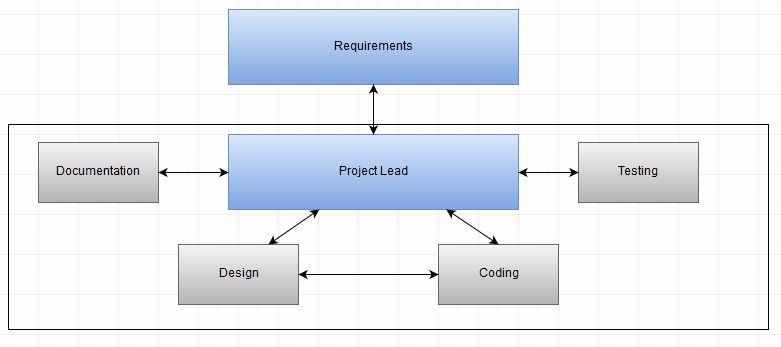


Figure - Group 3 Project Team

* + 1. Roles and Responsibilities

The project team consists of the following work roles and responsibilities.

* + - 1. Project Lead (Christy Gilliland)

The project lead coordinates with all team members to ensure the deliverable is completed on time and in accordance with all specifications. Furthermore, the project lead organizes all group chat sessions, phase reports and student-professor communications.

* + - 1. Requirements Lead (Patrick Smith)

Writes documentation for system and requirements specification. Assists with data needed for project plan and test plan.

* + - 1. Design Lead (Alexander Burch)

Responsible for the GUI design that brings all the games together. Assists with user guide, project plan, and testing. Collaborates with project and coding leads to establish interconnectivity between GUI and games.

* + - 1. Coding Lead (Patrick Smith)

Designs shared classes, game classes, and implements best practices for inheritance and reuse. Additionally, assists other developers to create a uniform appearance for the game suite. Serves as liaison between development team and other project work groups.

* + - 1. Testing Lead (Erik Freburger)

Creates, executes, and documents testing results of games in accordance with IEEE 829-1998, IEEE Standard for Software Test Documentation.

* + - 1. Documentation Lead (Christy Gilliland)

Overall in charge of gathering all project documentation. Works with other team leads to ensure all required documentation is gathered, organized and passed to project lead for inclusion in final report. Documentation lead is also responsible for the creation of the User’s Guide.

* 1. Managerial Process Plan

The managerial process plan emphasizes the work schedule and task management needed to meet the agreed upon project goals.

* + 1. Work Plan

The following section details both the work breakdown and project schedule for the group 3 project team.

* + - 1. Work Activities

The following table lists the breakdown of tasks according to its Software Development Life Cycle task.

| **Task name** | **Category** | **Subtask** |
| --- | --- | --- |
| Project Plan | Analysis | Business analysis |
| Test Plan | Design | Test Design |
| Project Design | Design | Software Analysis |
| Phase 1 Source | Code | Games Development |
| Phase 2 Source | Code | Games Development |
| Phase 3 Source | Code | Games Integration into Suite |
| Final Deliverable | Code | Software Build |

Table - Work Breakdown Schedule

* + - 1. Schedule Allocation

The figure below illustrates the project schedule in a Gantt chart view. This chart highlights the dependency mapping between tasks. The chart shows the major project milestones along with delivery due dates for each project deliverable.

Figure - Project Timeline

* + 1. Risk Management Plan

This sub clause of the SDP addresses the risks involved with the project. Its purpose is to identify, analyze, and prioritize risk factors. In addition, this sub clause will lay out specific risk mitigation plans. The probability of the risk occurring is graded on a 1 -5 scale with 1 being the lowest and 5 the highest probability of risk occurring.

| Risk ID | Description | Impact | Probability |
| --- | --- | --- | --- |
| 1 | All games are not completed by deadline | Inability to utilize games which were proposed in SRS | 1 |
| 2 | Games do not integrate smoothly into GUI Suite | Difficulty in utilizing all games seamlessly within the suite | 2 |
| 3 | Testing encounters problems | Incompletion of gaming suite | 2 |

Table - Risk Management Matrix

In order to mitigate the risk of project not meeting deadline group members will utilize on-line forums for communication in addition to a bi-weekly chat session. Secondly, the risk of games not properly integrating smoothly into the suite is mitigated by the use of GitHub for version control and coordination. Software modules can be checked out of GitHub and worked on by all team members allowing for seamless software integration. Lastly, testing will begin early and testers will be in constant communication with developers to quickly solve any issues which may arise.

* 1. Technical Process Plan

This sub clause of the SDP will be used to explain the process model being utilized as well as the tools, techniques, and methods to be used in the development of the software.

* + 1. Process Model

The process model to be used by the group 3 project team will be a traditional waterfall model for project management purposes. The developers have the ability to utilize either a standard waterfall model or an agile methodology within the allocated process timeline. The cross-functional integration of these two methodologies allows for flexible development of the software while maintaining an achievable set timeline for other tasks within the project.

* + 1. Methods, Tools, and Techniques

In order to provide for a uniform standard across the project team, the following tools will be utilized during project development:

* Language – Java
* Platform – JDK 6.0 or newer
* Code repository – GitHub
* Documentation – Microsoft Word and Excel

Coding will be in Java due to cross platform interoperability and its common usage in the development community. JDK 6.0 or newer will be used in order to utilize the most up-to-date features of the language. GitHub will serve as the main repository for both version control and accessibility. All documentation will be done in Microsoft Office, the industry standard office suite.

1. system specification

3.1. External Interface Requirements

3.1.1. User Interfaces

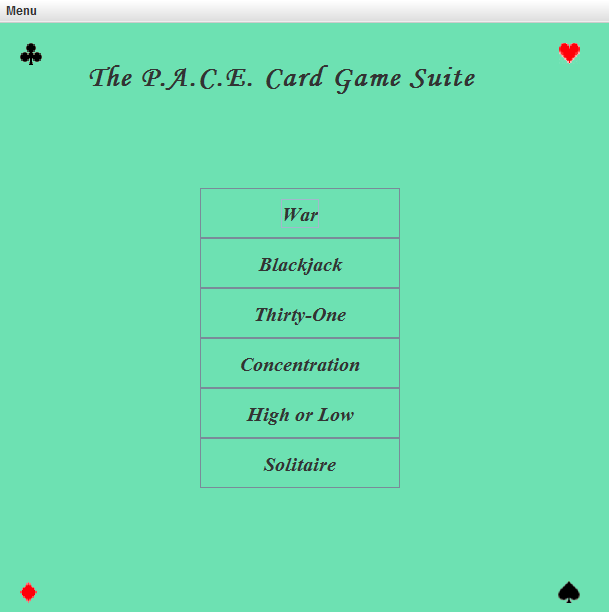
The system will provide a simple user interface. The user will access all games from the main screen and the system will transition to secondary screens to display the games.

Figure - Main User Interface

3.1.2. Hardware Interfaces

The system will require a mouse and a keyboard to select items on the screen and enter required information. The system must run Java SE 8. The system must have a minimum monitor size of 1280 by 1024 pixels.

1. requirements specification

4.1. Performance Requirements

4.1.1. Capacity

The system will only support one player at a time. Users will only play one game at a time. The save system will only be limited by available storage capacity.

4.2. Safety Requirements

No safety requirements identified at this time.

4.3. Security Requirements

Save files should not be human-readable to protect against cheating.

4.4. Dynamic Quality Requirements

* 99% of all games shall load in less than 30 seconds.
* 99% of all AI actions shall be processed in less than 2 seconds.
* 99% of all game status displays shall update in less than 1 second.
  1. Software Requirements

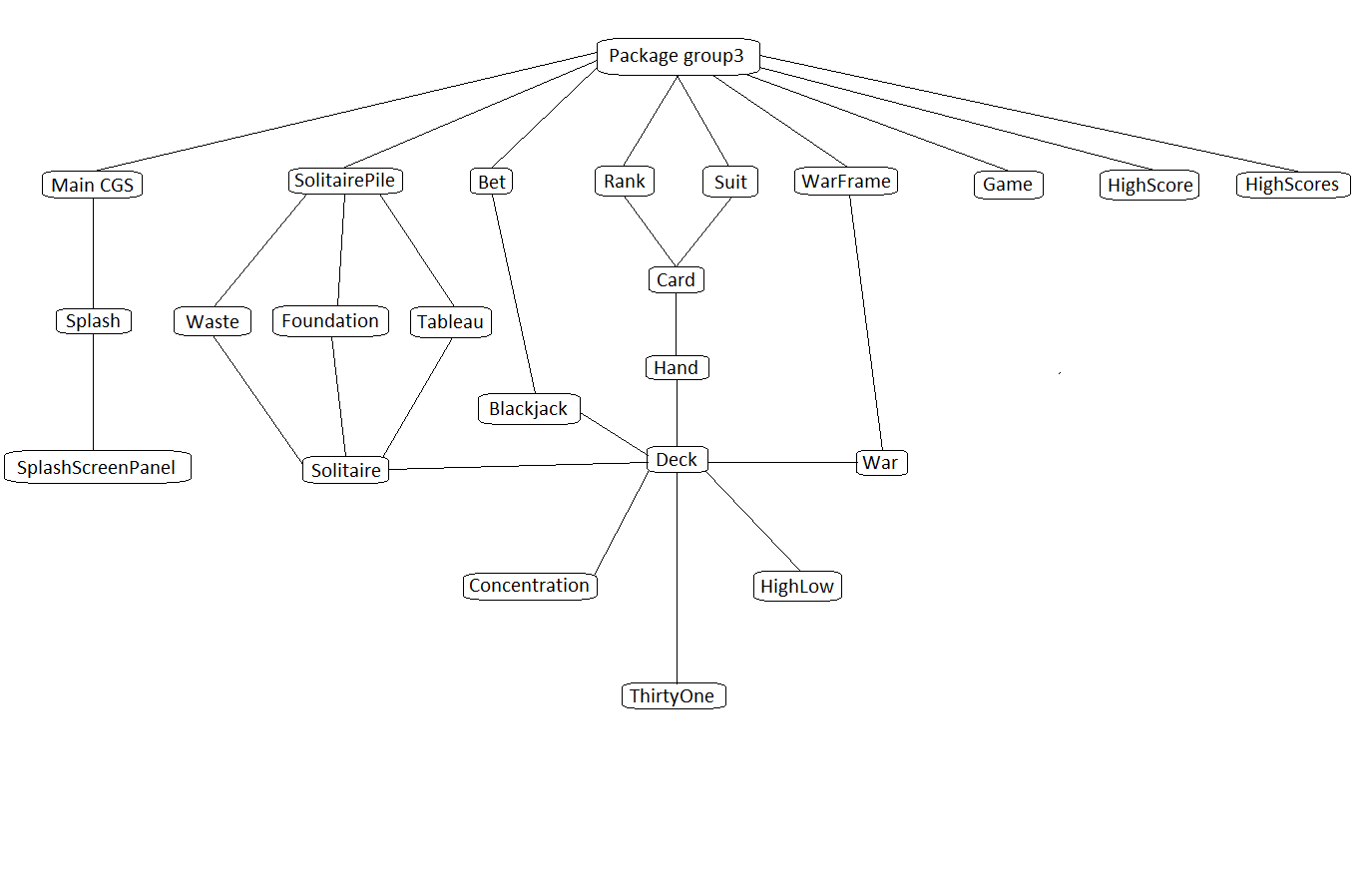


Figure - MainCGS UML Class Diagram

* + 1. Object: Card 
       1. Attributes
          1. Rank
          2. Suit
       2. Functions

4.5.1.2.1. Functional Requirement 1.1

The Card will provide the ability to query its state.



Figure - Example Card Object

* + - 1. Messages

None

* + - 1. Methods

| Method | Description |
| --- | --- |
| getBacks() | Returns image of back of card |
| Card(Rank rank, Suit suit) | Takes in a card's rank and suit and loads the correct image for the card |
| getRank() | Returns the card's rank |
| getSuit() | Returns the card's suit |
| flip() | Changes the card image from front to back and vice versa |
| writeObject(ObjectOutputStream out) | Serializes the Card |
| readObject(ObjectInputStream in) | Reads back in the Card from serialization |
| getPreferredSize() | Fits a card image to the best fit |
| toString() | Returns a string with the card's rank and suit |
| paintComponent(Graphics graphics) | Draws a graphic for the card |
| getBack() | Gets a BufferedImage of the Card back |
| getFront() | Gets a BuffereDImage of the Card front |

Table - Card Methods

* + 1. Object: Hand
       1. Attributes
          1. Cards
          2. Values
       2. Functions
          1. Functional Requirement 2.1

The Hand class will represent the cards the player has in a hand. Depending on the game, the player may have one or more hands. The values are assigned to cards based on the rules of the game for scoring purposes. Values should be initialized by each individual card game.

* + - * 1. Functional Requirement 2.2

The Hand should provide the ability to return the cards in the hand to a deck, get an individual card from a hand (view), remove a card (discard), and score a hand.

* + - 1. Messages

None

* + - 1. Methods

| Method | Description |
| --- | --- |
| addCard(Card card) | Adds a card to the hand |
| Resize() | Resizes the hand when the number of Cards in the hand changes. |
| scoreHand() | Totals the hand and checks to see if the total is above twenty one |
| isBusted() | Returns a boolean value depending on if the hand has a total greater than twenty one |
| toString() | Returns a String for the cards in the hand |
| getCards() | Returns a List of the Cards in the hand |
| getCard(int index) | Returns a card located at the given location |
| returnCards() | Removes the Cards in the hand and resizes the hand |
| handSize() | Returns the number of Cards in the hand |
| returnCard(Card card) | Removes a specific Card from the Hand |
| thirtyOneTotal() | Returns the total of a Hand with Aces worth 11 |
| total() | Returns the total value of the Cards in the hand |
| removeCard() | Removes the first Card from the hand and resizes the hand |
| isBlackjack() | Checks to see if a Hand has Blackjack |
| paintComponent(Graphics graphics) | Draws the hand based on the size of the hand |
| addCard(int index Card card) | Inserts the specified Card at the specified position in this hand. Shifts the Card currently at that position (if any) and any subsequent elements to the right (adds one to their indices). Used mostly for debugging Hands. |

Table - Hand Methods

* + 1. Object: Deck
       1. Attributes
          1. Cards
          2. Random Number Generator
       2. Functions
          1. Function Requirement 3.1

The deck shall provide the capability of shuffling itself randomly, dealing a card, and querying the remaining size of the deck.

* + - 1. Messages

None

* + - 1. Methods

| Method | Description |
| --- | --- |
| shuffle() | Shuffles the deck randomly |
| writeObject(ObjectOutputStream out) | Serializes the Deck |
| readObject(ObjectInputStream in) | Reads back in the Deck from serialization |
| ConcentrationShuffle() | Deals the deck for Concentration |
| cardFlip(int i) | Returns a card at a specified location |
| deal() | Deals a card from the deck |
| concentrationDeal(int i) | Gets a card from the deck without removing it |
| toString() | Returns a string of the Cards in Deck |
| deckSize() | Returns the number of Cards in Deck |
| paintComponent(Graphics graphics) | Draws the Deck |
| deckSize() | Returns the number of Cards in Deck |
| paintComponent(Graphics graphics) | Draws the Deck |

Table - Deck Methods

* + 1. Object: Game
       1. Attributes
          1. INITIAL\_LENGTH
       2. Functions
          1. Functional Requirement 4.1

Game must take in a String and loads or creates a high score file using that string as the name.

* + - * 1. Functional Requirement 4.2

Game must take in a file and an object (HighScores), and save the new high score to the file.

* + - * 1. Functional Requirement 4.3

Game must take in a file and try to load that file to return the object HighScores.

* + - * 1. Functional Requirement 4.4

Game must take in a maximum of three characters for a string from the user when the player sets a new high score and returns the string.

* + - 1. Messages

None

* + - 1. Methods

| Method | Description |
| --- | --- |
| loadOrCreateScores(String name) | If a high score exists, load it. If not, create it |
| saveHighScores(File scoreFile, HighScores scores) | Saves the high scores to a given file |
| loadHighScores(File scoreFile) | Loads the high scores from a given file |
| getInitials(Component frame) | Gets the player's initials for the high score records |

Table - Game Methods

* + 1. Object: HighScore
       1. Attributes
          1. score
          2. initials
       2. Functions
          1. Functional Requirement 5.1

HighScore must compare the current score to the current high score. If the current score beats the high score, it takes in the initials and saves it with the score.

* + - 1. Messages

None

* + - 1. Methods

| Method | Description |
| --- | --- |
| getScore() | Returns the score |
| getInitials() | Returns the initials |
| compareTo(HighScore h) | Compares to see if the score beats the current high score |
| toString() | Returns the initials and the score as a string |

Table - HighScore Methods

* + 1. Object: HighScores
       1. Attributes
          1. MAX\_SCORES
          2. highScores
       2. Functions
          1. Functional Requirement 6.1

HighScores must create a list of ten high scores

* + - * 1. Functional Requirement 6.2

HighScores must check to see if the current score is higher than the lowest high score and add it to the list.

* + - 1. Messages

None

* + - 1. Methods

| Method | Description |
| --- | --- |
| toString() | Formats the high scores as a string |
| isHighScore(double score) | Checks if a score qualifies as a high score |

Table - HighScores Methods

* + 1. Object: Bet
       1. Attributes
          1. bet
       2. Functions
          1. Functional Requirement 7.1

Bet must allow a player to bet a certain amount on the hand

* + - * 1. Functional Requirement 7.2

Bet must also pay out to the player if they win the hand

* + - 1. Messages

None

* + - 1. Methods

| Method | Description |
| --- | --- |
| getMoney() | Gets the money in the current bet |
| add(int bet) | Adds money to the current bet |
| payout(double payoutRatio) | Calculates how much the bet earned the player based on the payout ratio |

Table - Bet Methods

* + 1. Object: War
       1. Attributes
          1. AI hand
          2. Player hand
          3. Deck
       2. Functions
          1. Functional Requirement 4.1

War must provide the ability to play the game according to the game rules, and allow the game to be displayed to the screen. The game must expose a score and a status to the GUI. It must also provide an AI player.

* + - * 1. Functional Requirement 4.2

War must show the player a text or graphic based indicator of the relative sizes of the decks. The information must be clear enough to show the player who is winning the game.

* + - 1. Messages

After the game ends, the game object will generate a message that the GUI must display to the player.

* + - 1. Methods

| Method | Description |
| --- | --- |
| begin() | Starts a new War game |
| gui() | Creates a GUI for the War game |
| createMenuBar | Creates the JMenuBar and adds the menu items to it |
| reset() | Resets the game state for a new game |
| newGame() | Creates a new game session |
| battle() | Compares the top card of both players' Hands |
| checkGameOverConditions | Checks for win/loss conditions and displays the appropriate messages |
| war() | Returns a boolean value determining which player won in the case of a tie |
| updateHandSizeDisplay() | Updates the displayed Hand size for both players |
| directions() | Creates a JOptionPane stating the objectives of the game and how to play |
| highScores | Shows the high scores in a dialog box |
| saveGame() | Save the state of the game to a file |
| loadGame() | Load the game state from a file |

Table - War Methods

* + 1. Object: Concentration
       1. Attributes
          1. Deck
          2. Matches
          3. Failed Matches
       2. Functions
          1. Functional Requirement 5.1

Concentration must provide the ability to play the game according to the game rules, and allow the game to be displayed to the screen. The game must expose a score and a status to the GUI.

* + - * 1. Functional Requirement 5.2

Concentration must allow the player to see the cards before they are flipped back over when the player selects mismatched cards.

* + - 1. Messages

After the game ends, the game object will generate a message that the GUI must display to the player.

* + - 1. Methods

| Method | Description |
| --- | --- |
| newGame() | Creates a new game session |
| begin() | Starts the game play |
| actionPerformed(ActionEvent e) | Flips the card the user selects. It then waits for the user to select another card and runs checkForMatch() |
| checkForMatch() | Checks to see if the two cards flipped are a match. If so, the two are grayed out. Otherwise, they flip back over |
| gameWon() | States the player has won and uploads the score to high score if it is a new high score |
| createReferenceArray() | Sets components of the panel |
| createGUI() | Creates a GUI for the Concentration game |
| createMenu(JFrame theFrame) | Creates the menu and adds it to a menu |
| reset() | Resets the game |
| highScores() | Shows the high scores in a dialog box |
| mousePressed(MouseEvent e) | Checks to see if two cards are the same |
| loadGame() | Loads the game state from a file |
| Directions() | Shows the game play directions to the user |
| saveGame() | Saves the state of the game to a file |

Table - Concentration Methods

* + 1. Object: Blackjack
       1. Attributes
          1. dealer
          2. deck
          3. player
          4. playerBetHand1
          5. playerBetHand2
          6. playerSplitHand
       2. Functions
          1. Functional Requirement 10.1

Blackjack must provide the ability to play the game according to the game rules, and allow the game to be displayed to the screen. The game must expose a score and a status to the GUI. It must also provide an AI player for the dealer.

* + - * 1. Functional Requirement 10.2

Blackjack must allow the player to choose to hit, stand, split, or double down as appropriate.

* + - 1. Messages

After the game is over, the game object will generate a message that the GUI must display to the player.

* + - 1. Methods

| Method | Description |
| --- | --- |
| begin() | Runs newGame() and deal() |
| newGame() | Begins a new game |
| display() | Displays the value of the player's Hand |
| deal() | Deals out initial Hand in Blackjack. Shuffles the deck when the number of Cards left gets low |
| addCardFaceUp | Adds a card to the given player and flips it face up |
| updateRemain() | Updates the Cards remaining in the deck GUI element |
| hit() | Deals one card to the player |
| shiftToHandTwo() | Indicates the player is dealing with the second hand |
| stand() | Runs dealerPlays() |
| dealerPlays() | Dealer takes their turn |
| winQuestion() | Returns a JOptionPane stating the player won this hand and asks if they want to play again |
| loseQuestion() | Returns a JOptionPane stating the player lost this hand and asks if they want to play again |
| reset() | Removes the cards in play and runs deal() to start a new game |
| isSplittable | Returns a boolean value determining if the player has two cards of the same Rank |
| split() | Creates two separate hands for the player |
| isHandInPlay() | Returns a boolean value determining if the player has a hand in play |
| createGUI() | Creates a GUI for the Blackjack game |
| createBettingPanel() | Creates the betting panel to hold the bet denomination buttons |
| createButtons(int size) | Creates the action buttons |
| createMenu(JFrame theFrame) | Creates the menu and adds it to a menu |
| highScores() | Shows the high scores in a dialog box |
| Directions() | Creates a JOptionPane stating the objectives of the game and how to play |
| saveGame() | Saves the state of the game to a file |
| loadGame() | Loads the game state from a file |
| checkWinConditions() | Checks to see the result of the game. Either the player busted, the dealer busted, or one of the player's hands is higher or lower then the dealer's hand. Reports the state to the player and awards or discards the bet |

Table - Blackjack Methods

* + 1. Object: HighLow
       1. Attributes
          1. deck
          2. hand
       2. Functions
          1. Functional Requirement 7.1

HighLow must provide a way to compare the new card to the previous card. The user must be able to guess whether the next card will be higher or lower than the previous card.

* + - 1. Messages

After the game ends, the game object will generate a message that the GUI must display to the player.

* + - 1. Methods

| Method | Description |
| --- | --- |
| begin() | Starts the game play |
| newGame() | Creates a new game session |
| dealFlipped() | Deals a Card and flips it over |
| createGUI() | Creates a GUI for the HighLow game |
| checkGameOverConditions() | Checks for win/loss conditions and displays the appropriate messages |
| dealNextCard() | Deals the next Card to the discard spot |
| createMenu(JFrame frame) | Creates the menu and adds it to a menu |
| highScores() | Shows the high scores in a dialog box |
| reset() | Resets the game state for a new deck |
| loadGame() | Loads the game state from a file |
| saveGame() | Saves the state of a game to a file |

Table - HighLow Methods

* + 1. Object: ThirtyOne
       1. Attributes
          1. Deck
          2. Hand
          3. Center
       2. Functions
          1. Functional Requirement 8.1

ThirtyOne must provide the ability to play the game according to the game rules, and allow the game to be displayed to the screen. The game must expose a score and a status to the GUI. It must also provide an AI player for the player to play against.

* + - * 1. Functional Requirement 8.2

ThirtyOne must also allow the player to choose a card in the middle and choose which suits are positive and negative.

* + - 1. Messages

After the game ends, the game object will generate a message that the GUI must display to the player.

* + - 1. Methods

| Method | Description |
| --- | --- |
| begin() | Starts the game play |
| newGame() | Creates a new game and shuffles the deck |
| createGui() | Creates anew initial ThirtyOne GUI |
| createMenu(JFrame theFrame) | Creates the menu and adds it to a menu |
| highScores() | Shows the high scores in a dialog box |
| directions() | Shows the game play directions to the user |
| setCards() | Creates the buttons on the GUI and decides which player goes first |
| cardButtons() | Action listeners for each of the center buttons |
| saveGame() | Saves the state of the game to a file |
| loadGame() | Loads the game state from a file |
| reset() | Resets the game state for a new hand |
| results() | Decides which player wins, sees if the player wants to play again, and starts a new game or checks to see if they made the high score |
| switchCards() | Allows both players to discard a card to the center |

Table - ThirtyOne Methods

* + 1. Object: Solitaire
       1. Attributes
          1. deck
       2. Functions
          1. Functional Requirement 13.1

Solitaire must provide the ability to play the game according to the game rules, and allow the game to be displayed to the screen.

* + - * 1. Functional Requirement 13.2

Solitaire must allow the player to move cards onto different color cards with a rank one higher than the card in question or onto piles of the different suits.

#### 3.2.13.3. Messages

3.2.13.4 Methods

| Method | Description |
| --- | --- |
| createMenu(JFrame frame) | Creates the menu and adds it to a menu |
| newGame() | Creates a new game session |
| loadGame() | Loads the game state from a file |
| saveGame() | Saves the state of a game to a file |
| directions() | Shows the game play directions to the user |

Table - Solitaire Methods

* + 1. Object: Foundation
       1. Attributes
       2. Functions
          1. Functional Requirement 14.1

Foundation must get the top card from a deck and add or remove the card, if the move is valid.

* + - 1. Messages
      2. Methods

| Method | Description |
| --- | --- |
| isEmpty() | Checks to see if the arraylist cards is empty |
| getTopCard() | Returns the last Card in the arraylist cards |
| isValidMove(Card card) | Checks to see if the player can place a Card on top of another |
| addSingleCard(Card card) | Adds a single Card to the araylist cards |
| addCards(ArrayList<Card> c) | Adds multiple cards from the arraylist cards |
| removeSingleCard(Card card) | Removes the given Card from the arraylist cards |
| getAvailableCardsAt(Card card) | Returns an arraylist with the given card |
| paintComponent(Graphics graphics) | Repaints the Card, depending on if it is highlighted |
| removeCards(ArrayList<Card> cards) | Removes the Cards from the arraylist cards |

Table - Foundation Methods

* + 1. Object: SolitairePile
       1. Attributes
          1. Card
       2. Functions
          1. Functional Requirement 15.1

SolitairePile must get the top card from a deck and add or remove the card, if the move is valid.

* + - 1. Messages
      2. Methods

| Method | Description |
| --- | --- |
| Transform(Point pp, Point p) | Sets the point for a card to move to |

Table - SolitairePile Methods

* + 1. Object: Tableau
       1. Attributes
          1. deck
          2. card
       2. Functions
          1. Functional Requirement 16.1

Tableau needs to be able to support Solitaire by adding or removing cards, determining whether or not a move is valid, and repainting cards depending on whether or not they are highlighted.

* + - 1. Messages
      2. Methods

| Method | Description |
| --- | --- |
| Tableau(Deck deck, int numCards) | Sets the initial Cards into different piles |
| getAvailableCardsAt(Card card) | Returns an arraylist with the given card |
| isEmpty() | Checks to see if the arraylist cards is empty |
| getTopCard() | Returns the last Card in the arraylist cards |
| trimToSize() | Sets the Card width and height |
| addSingleCard(Card card) | Adds a single Card to the arraylist cards |
| removeCards(ArrayList<Card> c) | Removes the Cards from the arraylist cards |
| removeSingleCard(Card card) | Removes the given Card from the arraylist cards |
| isValidMove(Card card) | Checks to see if the player can place a Card on top of another |
| paintComponent(Graphics graphics) | Repaints the Card, depending on if it is highlighted |
| addCards(ArrayList<Card> cards) | Adds multiple Cards from the arraylist cards |

Table - Tableau Methods

* + 1. Object Waste
       1. Attributes
          1. deck
       2. Functions
          1. Functional Requirement 17.1

Waste must get the top card from a deck and add or remove the card, if the move is valid.

* + - 1. Messages
      2. Methods

| Method | Description |
| --- | --- |
| Waste(Deck deck) | Creates a new deck |
| paintComponent(Graphics graphics) | Repaints the Card |
| getTopCard() | Returns the last Card in the arraylist cards |
| addSingleCard(Card card) | Adds a single Card to the arraylist cards |
| removeSingleCard(Card card) | Removes the given Card from the arraylist cards |
| isValidMove(Card paramCard) | Checks to see if the player can place a Card on top of another |
| getAvailableCardsAt(Card card) | Returns an arraylist with the given card |
| addCards(ArrayList<Card> cards) | Adds multiple Cards from the arraylist cards |
| removeCards(ArrayList<Card> cards) | Removes the Cards from the arraylist cards |
| returnCards() | Removes all cards from the deck |

Table - Waste Methods

* + 1. Object: WarFrame
       1. Attributes
          1. warPanel
       2. Functions
          1. Functional Requirement 18.1

WarFrame must take in cards from both hands. The cards will then be displayed on the GUI.

* + - * 1. Functional Requirement 18.2

WarFrame must also calculate which player wins the battle. The winner will receive all of the cards offered up for the battle.

* + - 1. Messages

After the game ends, the game object will generate a message that the GUI must display to the player.

* + - 1. Methods

| Method | Description |
| --- | --- |
| setupGUI(Hand player, Hand ai, int warLevel) | Takes in both players' Hands and creates a GUI showing all of the cards given, along with who wins the war |
| returnCardsTi(Hand winner, Card[] playerWar, Card[] aiWar) | Returns the Cards won from the war to the winner |

Table - WarFrame Methods

* + 1. Object: Splash
       1. Attributes
          1. SplashScreenPanel
       2. Functions
          1. Functional Requirement 19.1

Splash must create a new SplashScreenPanel, display it for four seconds, and then dispose of the graphic.

* + - 1. Messages

None

* + - 1. Methods

| Method | Description |
| --- | --- |
| run() | Creates thread to draw new SplashScreenPanel |

Table - Splash Methods

* + 1. Object: SplashScreenPanel
       1. Attributes
          1. splashImage
       2. Functions
          1. Functional Requirement 20.1

SplashScreenPanel must be able to draw the given image.

* + - 1. Messages

None

* + - 1. Methods

| Method | Description |
| --- | --- |
| paintComponent() | Draws splashImage |

Table - SplashScreenPanel Methods

* + 1. Object: MainCGS
       1. Attributes
          1. War
          2. Concentration
          3. Blackjack
          4. HighLow
          5. ThirtyOne
          6. Solitaire
       2. Functions
          1. Functional Requirement 21.1

The GUI must tie all games together with a common interface. The GUI will expose a play surface to individual games. The games will place their game elements into this surface. The GUI must allow a way for the player to select a game.

* + - 1. Messages

None

* + - 1. Methods

| Method | Description |
| --- | --- |
| gui() | Creates main graphical user interface |
| createMenu(JFrame theFrame) | Creates the menu for the main card game suite |
| about() | Creates a new message pane stating the authors's names and each of the games' information |

Table - MainCGS Methods

1. design
   1. Product Perspective

The CGS will maintain scores in an external file for later recall. It will save data on game states into a save file (.ser extension).

Update

Player input

Load

Save

Request

Report

Key

Keyboard

Score Database

Click

Mouse

Save File

Display

Game state update

Figure - Data Flow Diagram Level 1

* 1. Product Functions

The CGS will offer the following functions: Initial Game Selection Menu, Game Load Menu, Game Save Menu, and Individual Game displays.

* + 1. Initial Game Selection Menu

This is the initial screen when the game loads. The game selection screen must display a clickable list of game options for the user to select. If a player selects the game option, the system must load a new instance of the game and hide the game selection menu. The menu must also present an option to load a previous game state when loading a selected game.

* + 1. Game Load Menu

The game will present the user with the option to load a saved game state.

* + 1. Game Save Menu

The user will have the option to save the game state at any time. The CGS will automatically save the game file to its parent directory as soon as the user click on “Save Game”.

* + 1. Individual Game Displays

Each game display will show the current state of the game graphically. It will allow the user to play the game using mouse or keyboard input. The game will follow the established rules generally accepted for the play of each individual game. The game will present an AI opponent if the game rules require one. The user will have the option to restart a game, save a game or load a game at any time. The game will display a message to the user showing when the user wins or loses a game.

* + 1. War

War will display a discard deck for the two cards in the war. There will be a war deck where subsequent cards are dealt in the event of a matching card war. The player and the computer will also have a deck. The game will track the number of wars won and the number of cards in each players’ deck visually.

* + 1. Concentration

Concentration will display four rows of 13 cards each. Each turn, the player will choose two cards. When the cards are chosen, they will turn face up. If the cards are the same rank and color, they are a match, and the player wins that pair and they can play again. If they are not a match, the game turns them over, and play passes to the computer or back to the player if the game is single player. The game is scored by either the number of matches found by each player or the number of incorrect pairs turned over for single player games.

* + 1. Blackjack

Blackjack will display a deck, the dealer’s hand, and the player’s hand. The dealer’s hand will show the first card dealt, and the second card will be displayed face down. The cards are dealt to the player and then the dealer, one at a time. The player is given the choice to hit, stand, double down, or split (if the two initial cards have the same value). If the player chooses to hit, they are dealt another card. If they stand, the play moves to the dealer. If they double down, they are allowed to increase the bet up to 100% and will receive only one more card. If they have two of the same card, the player can split. If they split, the player’s hand is separated into two separate hands with the same bet. The hands are treated separately, and the player plays both hands prior to the dealer playing his hand.

When play passes to the dealer, the dealer will display their card. The dealer will draw cards until their hand reaches 17 or higher. If the dealer goes over 21, they bust and the player wins. If the player goes over 21, the dealer does not draw, and the dealer wins. If neither bust, the player with the highest score 21 or less wins. The dealer wins all ties. A blackjack (an ace and a ten-point card) wins over a non-blackjack 21.

* + 1. High or Low

High or Low will display a deck and a face-up pile. There will be two buttons next to the deck, one marked “Lower!” and the other marked “Higher!” The player must guess whether the next card in the deck is lower or higher than the face-up card by clicking on the respective button. The game keeps a running score of the correct guesses.

* + 1. Thirty-One

Thirty-One will display the player’s hand and the center cards. All cards will be face up. The AI opponent’s hand is not shown. The player must make their choices by clicking on the cards they wish to pick up and discard. When a player thinks they may have the total closest to 31 in the same suit, they may knock. This allows the opponent to pick up/discard one more time before the round is scored.

* + 1. Solitaire

Solitaire will display the tableau (the seven piles that make up the main play area), the foundations (the four piles, one for each suit), the reserve deck and the waste pile. The player will move cards by clicking and dragging them to the desired location within game rules. The user wins the game if all four foundation piles are stacked from the ace to the king. Some games cannot be won.

* 1. User Characteristics

Users are assumed to have some experience with the card game they wish to play and little experience with computers. The system design and interfaces should be simple enough for users to interact with the system with no training.

* + 1. User Scenarios 
       1. User Scenario 1: start a game

1. User double clicks the CGS icon (jar file).
2. User selects a desired game.
3. The desired game loads.
4. The user plays the game.
   * + 1. User Scenario 2: load a game
5. User double clicks the CGS icon (jar file).
6. User selects a desired game.
7. User selects the load game function from the menu.
8. The previous game state loads.
9. User continues the game.
   * + 1. User Scenario 3: save a game
10. User double clicks the CGS icon (jar file).
11. User selects a desired game.
12. User plays the game.
13. User selects the save game function from the menu.
14. The CGS saves the current game state to its parent directory.
15. The game state is saved.
    1. Constraints

The system will follow the rules of the game. The system must prohibit users from performing illegal actions in each game.

The system must be free of errors and exploits, and faithfully recreate the experience of playing each game with actual cards.

* 1. Assumptions and Dependencies

Deployment of this system as described in this SRS assumes that a potential player has an existing computer that provides, at minimum:

* 500 MHz or faster processor
* 512 MB RAM
* 1 GB of hard drive space
* Windows XP, Vista, 7, 8.1, 10

1. user’s guide

Please see [Appendix A](#_APPENDIX).

1. test plan and results
   1. Test Plan Identifier

G3\_STS\_01

* 1. Objectives

The system test plan for the P.A.C.E Card Game Suite is aligned with the following objectives:

* State the purpose and goals of system testing
* Document the results of system testing
* Document the system testing plan
  1. Scope

The test plan is designed as a comprehensive set of system tests for all games within the game suite. Each individual game will be tested against an accepted standard set of rules for the specified game. Tests will include user interaction, game functionality and what if scenarios.

* 1. References (Applicable to the Software Test Plan)

The following references were used in creating this document.

* CMSC495 Group 3 Software Development Plan
* CMSC495 Group 3 System Requirements Specification
* CMSC495 Group 3 User guide
* IEEE. *IEEE Std 1233, 1998 IEEE Guide for Developing System Requirements Speciﬁcations.* IEEE Computer Society, 1998.
* IEEE. *IEEE Std 1058-1998 IEEE Recommended Practice for Software Design Descriptions.* IEEE Computer Society, 1998.
* IEEE. *IEE Std 829-1998 IEEE Standard for Software Test Documentation.* IEEE Computer Society, 1998.
  1. Architectural Context Diagram Mappings
     1. User Interface

The diagram below represents an abstract view of how the user interacts with the gaming suite. The gaming suite does not require any external interfaces besides a standard keyboard and mouse.

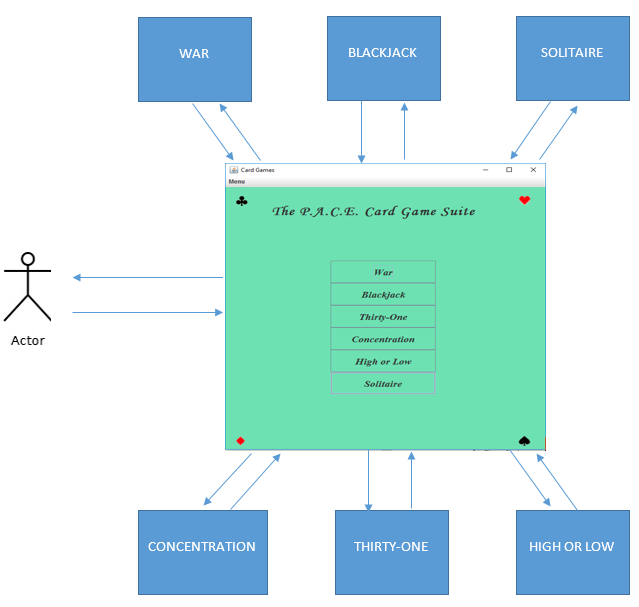


Figure - ACD of User Interface

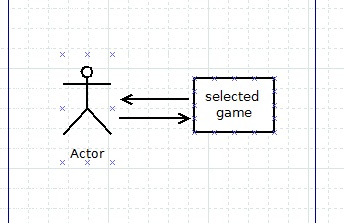


Figure - ACD of user interface with game

* 1. Test Items

All games which comprise the gaming suite will be tested during system testing. The gaming systems to be tested will consist of the following:

* Unified GUI interface
* War!
* Concentration
* Black Jack
* Thirty-one
* High or Low
* Solitaire
  + 1. Test Case Uniformity Standard
       1. Test Case ID

The following format will be utilized to document specific games to their corresponding test case id.

* Unified GUI interface – GUI\_xx
* WAR! – war\_xx
* Concentration – con\_xx
* Black Jack – bj\_xx
* Thirty – one – t1\_xx
* High or Low – hl\_xx
* Solitaire – sol\_xx
  + - 1. SRS Cross Reference

If applicable a SRS cross reference which can be mapped back to a specific section of the SRS will be included in the table. The following format will be utilized:

G3\_SRS\_<section number>

* + - 1. User Guide Reference

If applicable a user guide cross reference which can be mapped back to a specific section of the user guide will be included in the table. The following format will be utilized:

G3\_UG\_<heading>

* + - 1. Description

Description of test to be performed.

* + - 1. Expected Results

The expected results of each test.

* + - 1. Actual Results

The actual results of the test.

* + - 1. Pass / Fail

Whether each test has passed or failed based on the test results. It is important to note the results of the test do not have to exactly correspond to the expected results of each test.

* + - 1. Remarks

Tester remarks will go here.

* 1. Environmental Needs
     1. Hardware

The gaming suite is being built utilizing the Java programming language which has the capability to run on any platform. All system testing will be performed on a HP laptop utilizing AMD A10-460M 64 bit 2.30 GHz processor with 6.00 GB of RAM.

* + 1. Software

All games systems will be tested on a Windows 10, 64-bit operating system.

* 1. User Test Cases

Each game along with Unified GUI has its own test case specification table. The test case specification table. The test case table will utilize the following format:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Cross Reference** | **Description** | **Expected Results** | **Actual Results** | **Pass / Fail** | **Remarks** |
|  |  |  |  |  |  |  |

Table - Test Case Template

* + 1. Unified GUI

The below table represents the test cases for the unified GUI.

| **Test Case ID** | **Cross Reference** | **Description** | **Expected Results** | **Actual Results** | **Pass / Fail** | **Remarks** |
| --- | --- | --- | --- | --- | --- | --- |
| GUI\_01 | G3\_SRS\_2.2.1 | Double left click of icon will open Unified GUI screen. | Unified interface  Displays. All games displayed | double click opened Unified GUI | pass | See App B Figure 1 |
| GUI\_02 | N/A | Hover mouse of specific game | Moving mouse to each game will highlight the game | each game is highlighted | pass | n/a |
| GUI\_03 | G3\_SRS\_2.2.1 | Open specified game by double left click | specified game opens | specified game opened | pass | n/a |
| GUI\_03 | N/A | click of left mouse button on minimize window icon | display disappears from desktop and is shown only on taskbar | display disappeared from desktop and is shown only on taskbar | pass | n/a |
| GUI\_04 | N/A | click of left mouse button on window restore down / maximize button | window becomes full or small size depending on current state | window transforms to proper state depending on initial state | pass | n/a |
| GUI\_05 | N/A | left mouse click of **x** button closes window | window closes and icon disappears in taskbar | window closed and icon disappeared in taskbar | pass | n/a |

Table - Unified GUI Test Table

* + 1. War!

The below table represents the test cases for War!

| **Test Case ID** | **Cross Reference** | **Description** | **Expected Results** | **Actual Results** | **Pass / Fail** | **Remarks** |
| --- | --- | --- | --- | --- | --- | --- |
| WAR\_01 | G3\_SRS\_2.2.2 | single left mouse click opens war! Game. | game opens | game opens | pass | See App B Figure 2 |
| WAR\_02 | G3\_SRS\_2.2.2 | select load game | previous games in correct state open | previous game opens | pass | n/a |
| WAR\_03 | G3\_SRS\_2.2.2 | select new game | new game is opened for player | new game opens | pass | See App B Figure 2 |
| WAR\_04 | G3\_SRS\_2.2.2.4.1 | player deck, and AI deck displayed | all card decks are displayed on screen | all decks shown | pass | See App B Figure 2 |
| WAR\_05 | G3\_SRS\_2.2.2.4.1 | number of cards in each player’s deck is shown | proper number of cards displayed for each player | number cards displayed | pass | See App B Figure 2 |
| WAR\_06 | G3\_SRS\_2.2.2.4.1 | At end of game if player has high score pop-up window displays with prompt to enter initials | pop up displays | high score pop-up appears in game | pass | n/a |
| WAR\_07 | G3\_UG\_WAR | click “battle” button to play game | game begins or continues depending on current state | game continues | pass | See App B Figure 2 |
| WAR\_8 | G3\_UG\_WAR | higher card wins after battle button is pressed | The player with higher card keeps wins the round | player with higher card wins | pass | See App B Figure 3 |
| WAR\_9 | G3\_UG\_WAR | both player’s cards displayed are of equal value | 3 cards are placed face down with a fourth displayed. Higher card wins all cards | higher card displayed wins | pass | See App B  Figure 4 |

Table - War! Test Table

* + 1. Concentration

The below table represents the test cases for Concentration.

| **Test Case ID** | **Cross Reference** | **Description** | **Expected Results** | **Actual Results** | **Pass / Fail** | **Remarks** |
| --- | --- | --- | --- | --- | --- | --- |
| CON\_01 | G3\_SRS\_2.2.2 | single left mouse click opens concentration Game. | concentration opens | concentration opens | pass | n/a |
| CON\_02 | G3\_SRS\_2.2.2 | select “save game” | saves game in correct state | game saves | pass | See App B  Figure 5 |
| CON\_03 | G3\_SRS\_2.2.2 | open new game | new game is opened for player | new game opens | pass | n/a |
| CON\_04 | G3\_SRS\_2.2.4.2 G3\_UG\_CONCENTRATION | display five rows of 6 cards each | five rows of 6 cards displayed | five rows of 6 cards displayed | pass | See App B  Figure 5 |
| CON\_05 | G3\_SRS\_2.2.4.2 G3\_UG\_CONCENTRATION | Select two cards and if they match | cards are grayed out and play continues | when cards match they are grayed out | pass | See App B  Figure 5 |
| CON\_06 | G3\_SRS\_2.2.4.2 | select two cards and no match | both cards return to face down position and player can try again | both cards return to face down | pass | n/a |
| CON\_07 | G3\_SRS\_2.2.4.2 G3\_UG\_CONCENTRATION | game scoring determined by number of incorrect guesses | The score will change based on number of incorrect guesses. Lower score is a better played game | Score is displayed as the number of attempts and includes the time it took to play the game. | pass | See App B  Figure 5 |
| CON\_08 | G3\_SRS\_2.2.4.2 | matched all cards | message box displays with score | score displayed | pass | n/a |

Table - Concentration Test Table

* + 1. Black Jack

The below table represents the test cases for Black Jack.

| **Test Case ID** | **Cross Reference** | **Description** | **Expected Results** | **Actual Results** | **Pass / Fail** | **Remarks** |
| --- | --- | --- | --- | --- | --- | --- |
| BJ\_01 | G3\_SRS\_2.2.2 | single left mouse click opens Black Jack Game. | game screen opens with new game | game screen opens | pass | n/a |
| BJ\_02 | G3\_SRS\_2.2.2 | select “load game” | saved game in correct state opens | Hands from previous game does not load | fail | see BJ\_02\_1 |
| BJ\_02\_1 | G3\_SRS\_2.2.2 | select “load game” | saved game in correct state opens | Game loads correctly | pass | retest for BJ\_02 |
| BJ\_03 | G3\_SRS\_2.2.2 | open new game | new game is opened for player | new game opens | pass | n/a |
| BJ\_04 | G3\_SRS\_2.2.3 | option to save game in menu | game saves | Game saves | pass | validated by using game load |
| BJ\_05 | G3\_SRS\_2.2.4.3 | game screen initial display | display a deck, the dealer’s hand, and the player’s hand | everything displayed | pass | n/a |
| BJ\_06 | G3\_SRS\_2.2.4.3 | play is started | The cards are dealt to the player and then the dealer, one at a time | cards dealt one at time in correct order | pass | n/a |
| BJ\_07 | G3\_SRS\_2.2.4.3 | select hit | player is dealt another card | player gets another card | pass | n/a |
| BJ\_08 | G3\_SRS\_2.2.4.3 | select stand | play moves to dealer | play moves to dealer | pass | n/a |
| BJ\_09 | G3\_SRS\_2.2.4.3 | select double down | allowed to increase the bet up to 100% and will receive only one more card | allowed to increase the bet up to 100% and will receive only one more card | pass | n/a |
| BJ\_10 | G3\_SRS\_2.2.4.3 | select split | two separate hands with same bet | two separate hands with same bet | pass | n/a |
| BJ\_11 | G3\_SRS\_2.2.4.3 | player stands | play is passed to dealer | play passed to dealer | pass | See App B Figure 6 |
| BJ\_12 | G3\_SRS\_2.2.4.3 | player’s hand goes over 21 | bust and dealer wins | bust | pass | n/a |
| BJ\_13 | G3\_SRS\_2.2.4.3 | play is passed to dealer | dealer will play cards until 17 or higher | dealer plays to 17 or higher | pass | See App B Figure 6 |
| BJ\_14 | G3\_SRS\_2.2.4.3 | dealer hand goes over 21 | bust and player wins | bust | pass | n/a |
| BJ\_15 | G3\_SRS\_2.2.4.3 | neither player busts | player with 21 or closest score to wins | player closest wins | pass | See App B Figure 6 |
| BJ\_16 | G3\_SRS\_2.2.4.3 | tie | dealer wins all ties | dealer wins | pass | n/a |

Table - Blackjack Test Table

* + 1. Thirty-One

The below table represents the test cases for Thirty-One.

| **Test Case ID** | **Cross Reference** | **Description** | **Expected Results** | **Actual Results** | **Pass / Fail** | **Remarks** |
| --- | --- | --- | --- | --- | --- | --- |
| T1\_01 | G3\_SRS\_2.2.2 | single left mouse click opens thirty-one Game. | new game opens | new game opens | pass | n/a |
| T1\_02 | G3\_SRS\_2.2.2 | select saved game | previous games in correct state open | game saves | pass | test validated by using “load game” feature |
| T1\_03 | G3\_SRS\_2.2.2 | open new game | new game is opened for player | new game loads | pass | n/a |
| T1\_04 | G3\_UG\_THIRTY-ONE | game screen initial display | player’s cards and center cards displayed | player’s cards and center cards displayed | pass | n/a |
| T1\_05 | G3\_UG\_THIRTY-ONE | game play initialized | player and AI are each dealt 3 cards | each player dealt three | pass | n/a |
| T1\_06 | G3\_UG\_THIRTY-ONE | player chooses from center deck | card player chooses from center will switch with card in hand that the player chooses | card player chose from center switched with card in hand that the player chose | pass | See App B Figure 8 |
| T1\_07 | G3\_UG\_THIRTY-ONE | game is scored | Player with 31 or closest to 31 wins | pop up announces if player won or lost and score | pass | See App B Figure 9 |

Table - Thirty-One Test Table

* + 1. High or Low

The below table represents the test cases for High or Low.

| **Test Case ID** | **Cross Reference** | **Description** | **Expected Results** | **Actual Results** | **Pass / Fail** | **Remarks** |
| --- | --- | --- | --- | --- | --- | --- |
| HL\_01 | G3\_SRS\_2.2.2 | single left mouse click opens high low Game. | game opens | game opens | pass | n/a |
| HL\_02 | G3\_SRS\_2.2.2 | select load game | previous games in correct state open | saved game opens | pass | See App B  Figure 7 |
| HL\_03 | G3\_SRS\_2.2.2 | open new game | new game is opened for player | new game opens | pass | n/a |
| HL\_04 | G3\_UG\_HIGHLOW | select lower | if card hidden on deck lower than card displayed point to player if not no point | if lower point awarded if higher no point | pass | n/a |
| HL\_05 | G3\_UG\_HIGHLOW | select higher | if card hidden on deck higher than card displayed point to player if not no point | if higher point awarded if not no point | pass | n/a |
| HL\_06 | G3\_UG\_HIGHLOW | display score | proper score for game is displayed | score displayed properly | pass | See App B  Figure 7 |
| HL\_07 | G3\_UG\_HIGHLOW | 52 cards in deck | during game play 52 cards are used | all cards used | pass | n/a |
| HL\_08 | G3\_UG\_HIGHLOW | verify rank of cards from high to low K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2, A | Cards will be denoted in proper order during game play | card ranking correct | pass | n/a |

Table - High or Low Test Table

* + 1. Solitaire

The below table represents the test cases for Solitaire.

| **Test Case ID** | **Cross Reference** | **Description** | **Expected Results** | **Actual Results** | **Pass / Fail** | **Remarks** |
| --- | --- | --- | --- | --- | --- | --- |
| SOL\_01 | G3\_SRS\_2.2.2 | single left mouse click opens high low Game. | game opens | game opens | pass | n/a |
| SOL\_02 | G3\_SRS\_2.2.2 | select load game | previous games in correct state open | saved game opens | pass | n/a |
| SOL\_03 | G3\_SRS\_2.2.2 | open new game | new game is opened for player | new game opens | pass | n/a |
| SOL\_04 | G3\_UG\_SOLITAIRE | move an ace to foundation pile by clicking on it then click on a foundation pile | ace moves | ace moves | pass | See App B Figure 10 |
| SOL\_05 | G3\_UG\_SOLITAIRE | move a group of cards to another tableau pile by clicking on it then click on the destination pile | pile moves | pile moves | pass | Cards must be alternating colors and next rank order card.  See App B Figure 11 |
| SOL\_06 | G3\_UG\_SOLITAIRE | reserve deck should flip over and “restart” when player gets to the end of it | deck flips over | deck flips over | pass | n/a |
| SOL\_07 | G3\_UG\_SOLITAIRE | some games  cannot be finished; no more moves can be made to advance the game | game is still functional, but cannot be finished | game is still functional, but cannot be finished | pass | the game does not generate a message, which is expected in a traditional Solitaire game. Player must select “new game”.  See App B Figure 12 |
| SOL\_08 | G3\_UG\_SOLITAIRE | Game is complete when all cards have been arranged in the four foundation piles | 4 foundation piles from Aces to Kings; all other piles empty | 4 foundation piles from Aces to Kings; all other piles are empty | pass | See App B Figure 13 |

Table - Solitaire Test Table

* 1. Test Deliverables

Test deliverables will consist of the following:

* Software Test Specification (STS)
* Test tables
* Screenshots
  + 1. Testing Tables

The test tables which are described in the STS will be incorporated into the test deliverables.

1. development history

| **Deliverable** | **Due Date** | **Status** |
| --- | --- | --- |
| Project Plan | 9/4/2016 | Completed |
| Test Plan | 9/11/2016 | Completed |
| Test Plan / Testing | 10/09/2016 | In progress |
| User’s Guide | 9/11/2016 | Completed |
| Project Design | 9/18/2016 | Completed |
| Phase I Report + Source Code | 9/25/2016 | Completed |
| Fully functioning/commented code (5 Original Games) | 9/27/2016 | Sprint method started 08/24/2016. Completed  (Date set by team) |
| Fully functioning/commented code (Solitaire) | 10/2/2016 | Completed |
| Phase 2 Report + Source Code | 10/2/2016 | Completed |
| Phase 3 Report + Source Code | 10/9/2016 | Completed |
| Final Project | 10/10/2016 | Completed |

1. conclusions
   1. Lessons Learned

A list of skills, tips, and tricks we learned:

* Opening a pdf file from within a Java class
* Using images for buttons and labels
* Extending JPanels instead of creating new ones
* Creating menus
* Loading and saving files
* Manually tagging titles in a Word document that contains more than one TOC so Word can determine which entries go in which TOC
* Creating new labels in a Word document that contains more than one TOF so Word can determine which entries go in which TOF
  1. Design Strengths

The P.A.C.E Card Game Suite maximizes code reusability with custom designed classes, such as deck, hand, card, etc. It insures that all games use the same graphics, design, and card values.

Each of the games has the capability of creating its own save file. Each game uses a proprietary save file to prevent confusion. The P.A.C.E Card Game Suite allow more than one game to run at the same time.

* 1. Limitations
  2. Suggestions for Future Improvement

With more time, group 3 would be interested in adding more games and including animations, more custom graphics, and avatars.

## APPENDIX

[Return to Main Document](#Users_Guide)

## 



**User’s Guide**

Version 2.0

**Table of Contents**

[Return to Main Document](#Users_Guide)

[ABOUT P.A.C.E CARD GAME SUITE A-4](#_Toc463713885)

[MINIMUM SYSTEM REQUIREMENTS A-4](#_Toc463713886)

[INSTALLATION AND SETUP A-4](#_Toc463713887)

[CONTRIBUTORS A-4](#_Toc463713888)

[P.A.C.E CARD GAME SUITE MAIN MENU A-5](#_Toc463713889)

[THE CARD GAMES A-5](#_Toc463713890)

[WAR A-6](#_Toc463713891)

[*The Objective* A-6](#_Toc463713892)

[*How to Play* A-6](#_Toc463713893)

[BLACKJACK A-8](#_Toc463713894)

[*The Objective* A-8](#_Toc463713895)

[*How to Play* A-8](#_Toc463713896)

[CONCENTRATION A-10](#_Toc463713897)

[*The Objective* A-10](#_Toc463713898)

[*How to Play* A-10](#_Toc463713899)

[HIGH OR LOW A-11](#_Toc463713900)

[*The Objective* A-11](#_Toc463713901)

[*How to Play* A-11](#_Toc463713902)

[THIRTY-ONE A-12](#_Toc463713903)

[*The Objective* A-12](#_Toc463713904)

[*How to Play* A-12](#_Toc463713905)

[SOLITAIRE A-13](#_Toc463713906)

[*The Objective* A-13](#_Toc463713907)

[*How to Play* A-13](#_Toc463713908)

[IN-GAME MENUS A-14](#_Toc463713909)

[Menu Options A-14](#_Toc463713910)

[*New Game* A-14](#_Toc463713911)

[*Save Game* A-14](#_Toc463713912)

[*Load Game* A-14](#_Toc463713913)

[*How to Play* A-14](#_Toc463713914)

[*High Scores* A-15](#_Toc463713915)

[*Return to main menu* A-15](#_Toc463713916)

[CONTACT US A-15](#_Toc463713917)

**Table of Figures**

[Return to Main Document](#Users_Guide)

[Figure 1 - P.A.C.E Card Game Suite Main Menu A-5](#_Toc463713918)

[Figure 2 - War Battles A-6](#_Toc463713919)

[Figure 3 - It's War! A-7](#_Toc463713920)

[Figure 4 - The Blackjack Table A-9](#_Toc463713921)

[Figure 5 - Concentration Board A-10](#_Toc463713922)

[Figure 6 - High or Low A-11](#_Toc463713923)

[Figure 7 - Thirty-One Table A-12](#_Toc463713924)

[Figure 8 - Solitaire Table A-13](#_Toc463713925)

[Figure 9 - In-Game Menu A-14](#_Toc463713926)

ABOUT P.A.C.E CARD GAME SUITE

[Return to Main Document](#Users_Guide)

P.A.C.E Card Game Suite is a compilation of classic card games sure to provide many hours of entertainment! The suite contains the full versions of War, Concentration, Black Jack, High or Low, Thirty-One, and Solitaire.

MINIMUM SYSTEM REQUIREMENTS

|  |  |
| --- | --- |
| Operating System: | Windows 7 / 8 / 10, 64-bit |
| CPU: | Celeron (2 GHz) or better |
| Software: | Java SE Runtime Environment (8u101 or later) |
| Memory | 65 KB for game file; 10 KB for High Scores file |
| Input/Output: | Mouse (required), keyboard (optional) |

INSTALLATION AND SETUP

The P.A.C.E Card Game Suite is delivered as an executable Java (Jar) file. There is no installation required – just click on the game to get started!

CONTRIBUTORS

GROUP 3:

Alexander Burch – Requirements Lead, Coding

Christy Gilliland – Project and Documentation Lead, Coding

Erik Freburger – Testing Lead

Patrick Smith – Coding and Requirements Lead

P.A.C.E CARD GAME SUITE MAIN MENU

[Return to Main Document](#Users_Guide)

The main menu contains a list of games available in the P.A.C.E Card Game Suite. Click on the game you want to play and a new game window will open. When you are ready to play a different game, simply close the game window and you will be returned to the Main Menu!



Figure - P.A.C.E Card Game Suite Main Menu

THE CARD GAMES

The P.A.C.E Card Game Suite contains the following games:

* War
* Black Jack
* Concentration
* High or Low
* Thirty-One
* Solitaire.

Each of the games has its own customized window and menus.

WAR

[Return to Main Document](#Users_Guide)

*The Objective*

The objective of War is to get all 52 cards by winning battles against your opponent. In P.A.C.E Card Game Studio, your AI opponent is customized to be a random player.

*How to Play*

Card Rank, from the highest to the lowest: K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2, Ace

The deck is split evenly between you and your opponent. Both players will play the top card in their deck at the same time (in P.A.C.E, all you have to do is click the “Battle” button to play your cards and your opponent’s cards). If your card is higher than your opponent’s card, you win that match and the two cards are automatically returned to the bottom of your deck. If you lose, your opponent gets to keep the cards!

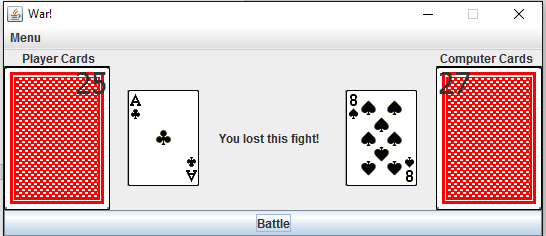


Figure - War Battles

In the event of a “tie”, a War is declared! Both players will then play the next three cards in their deck (face down) and a fourth card (known as the “Draw!”) is played face up. If your Draw card is higher than your opponent’s Draw card, you won the battle and you get to keep all of the cards in play. If it is not, your opponent keeps all of the cards.

[Return to Main Document](#Users_Guide)

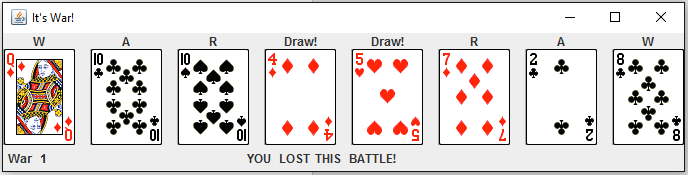


Figure - It's War!

In the event of a tie, a War Battle is repeated. If War is declared and a player does not have enough cards to battle with, the game will automatically determine how many cards are left and customize the number of cards drawn.

When one player succeeds in getting all 52 cards, they win!

BLACKJACK

[Return to Main Document](#Users_Guide)

*The Objective*

The objective of Blackjack is to beat the dealer by getting the highest scoring hand without exceeding 21. In P.A.C.E Card Game Studio, your AI dealer is customized to be a random player.

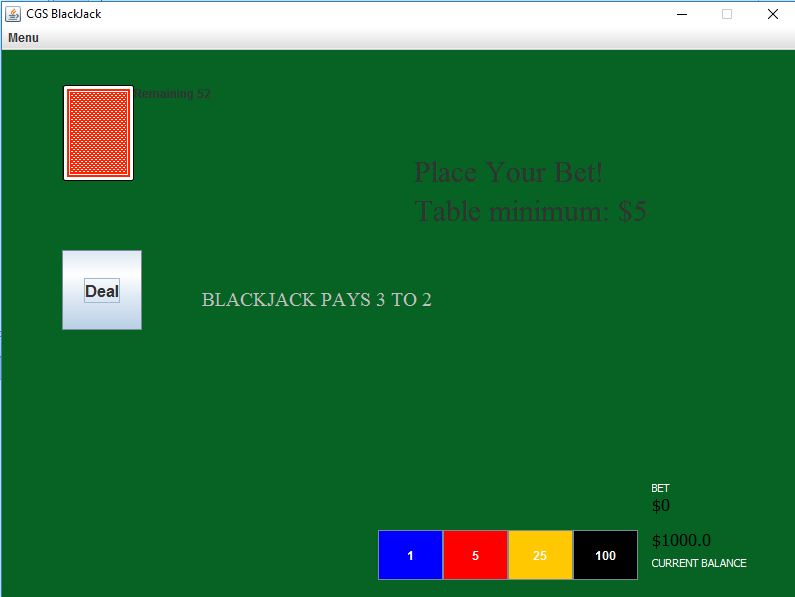
*How to Play*

Card Values: Aces can have a value of either one or eleven. Face cards are worth 10, and all other cards are worth their face value.

You will start with a $1,000 bank balance for each new game. Possible bet amounts are: $1, $5, $25, and $100.

The dealer and the player will be dealt two cards: one up, one down. The player goes first. If you choose to “Hit”, you will receive another card. Continue to do this until you get 21 (or as close as possible without going over 21). If the value of your cards exceeds 21, it is a bust and the hand is over. If you choose to “Stand”, the dealer must hit if the dealers hand value is less than 17. In the case of a tie, the dealer wins.

In P.A.C.E Black Jack, you have the option to “split” your hand if you are initially dealt two of the same cards. Your bet is doubled and you will receive two more cards to form two separate hands. The basic rules still apply for each hand.



[Return to Main Document](#Users_Guide)

Figure - The Blackjack Table

If your initial hand has a ten-card and an ace, and the dealer’s hand does not, you automatically win the hand! This is called “Blackjack”.

CONCENTRATION

[Return to Main Document](#Users_Guide)

*The Objective*

The object of Concentration is to turn over pairs of matching cards. There is no AI opponent in a game of Concentration.

*How to Play*

Concentration is played with 30 cards (15 pairs). To play, click on any two cards. If the two cards match, they will be highlighted in gray. If they do not, they will flip back over. Continue until you have matched all 15 pairs. See if you can beat your high score by matching all pairs in the least number of moves!

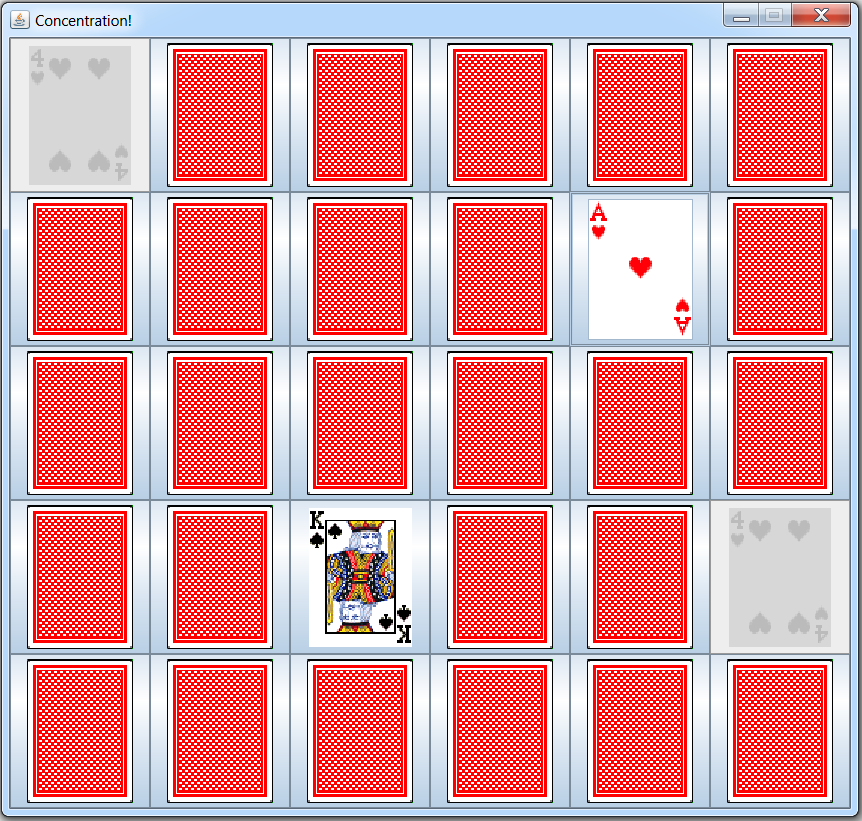


Figure - Concentration Board

HIGH OR LOW

[Return to Main Document](#Users_Guide)

*The Objective*

The objective of High or Low is to guess whether the next card is higher or lower than the current face up card. There is no AI opponent in a game of High or Low.

*How to Play*

Card Rank, from the highest to the lowest: Ace, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2

High or Low is played with 52 cards. Guess whether the next card is higher or lower than the current face up card. If the next card is equal to the current card, it is counted as a wild and you get the point! Can you score a perfect 51?

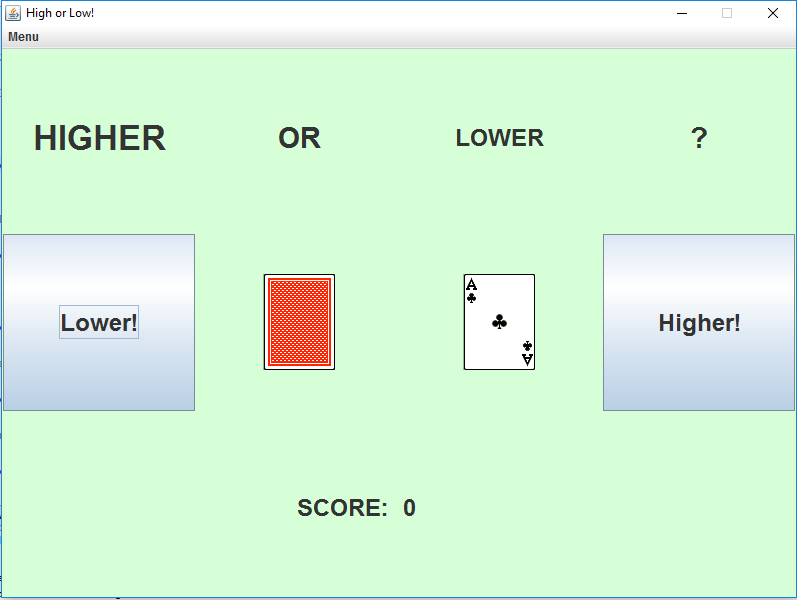


Figure - High or Low

THIRTY-ONE

[Return to Main Document](#Users_Guide)

*The Objective*

The objective of Thirty-One is to obtain a hand with a value as close as possible to 31 in the same suit. In P.A.C.E Card Game Studio, your AI opponent is customized to be a random player.

*How to Play*

Card values: Aces are worth 11, face cards are worth 10, and all other cards are worth their face value.

Both players are dealt a hand of 3 cards. When it is your turn, you may choose to pick up from either the stock pile or the discard pile, and then choose a card to discard. At the start of any turn, if you believe you have a winning hand, you may choose to “knock” instead of drawing a card. Your opponent will then have one more chance at picking up and discarding before the hand is scored.

If you have 31, or the closest to 31 without going over, in the same suit…you win!

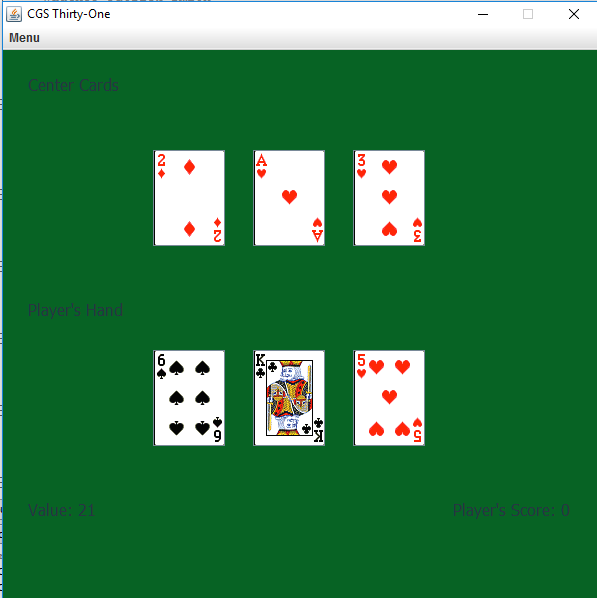


Figure - Thirty-One Table

SOLITAIRE

[Return to Main Document](#Users_Guide)

*The Objective*

The objective of Solitaire is build four foundation piles: one for each suit, from the lowest to the highest. In P.A.C.E Card Game Studio, Solitaire is a single player game.

*How to Play*

Card Rank, from the highest to the lowest: K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2, Ace

P.A.C.E Card Game Studio automatically sets up the same as shown below:

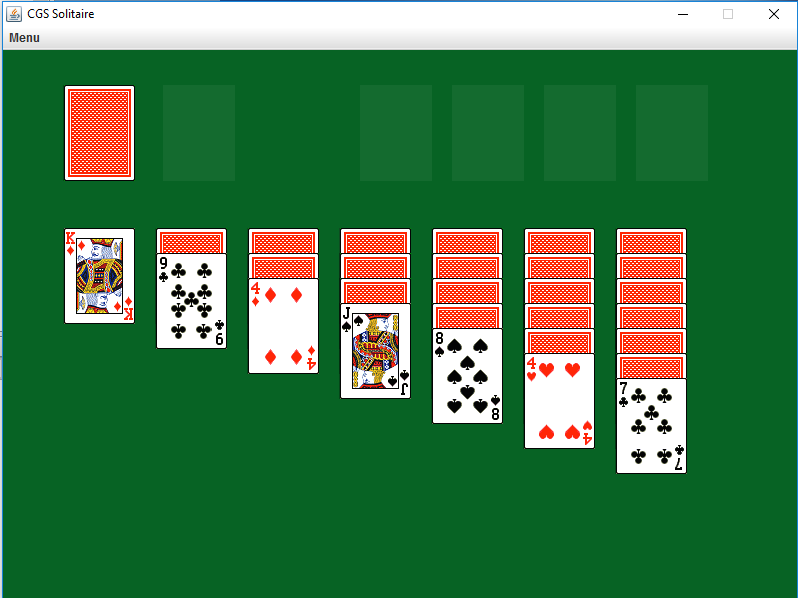


Figure - Solitaire Table

You may start the foundation piles as soon as you uncover the ace of a suit. Continue rearranging the cards, moving only the face up cards. To place a card on top, it must be a different color than the card you are placing it on top of and have a value of one less. As play continues, you may move a card to a foundation pile if it is the next card (in rank order). When you run out of moves, you can use the reserve deck. Three cards will be dealt from the reserve deck and you must use them in order.

[Return to Main Document](#Users_Guide)

Build all four foundation piles to win the game!

IN-GAME MENUS

[Return to Main Document](#Users_Guide)

Every game in the P.A.C.E Card Game Suite has the same menu.

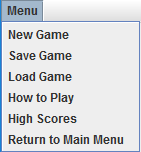


Figure - In-Game Menu

Menu Options

*New Game*

It’s simple enough! Click on “New Game” in any game window to start a new game.

*Save Game*

Clicking on “Save Game” will save your current game. Keep in mind that you can only have one saved state for each game in the P.A.C.E Card Game Suite.

*Load Game*

Clicking on “Load Game” will load the most recently save game. Keep in mind that you can only have one saved state for each game in the P.A.C.E Card Game Suite.

*How to Play*

Clicking on “How to Play” will display the basic rules of the game you are currently playing.

[Return to Main Document](#Users_Guide)

*High Scores*

Clicking on “High Scores” will display the list of your high scores for the game you are currently playing. To see your high scores for another game, access this menu option within that game window.

*Return to main menu*

Clicking on “Return to main menu” will close the current game screen and return to the main menu. Clicking on the “X” in the upper right corner will also close the current game screen and return to the main menu.

CONTACT US

For more information or to report a bug, please contact us at [PACE@notarealemail.com](mailto:PACE@notarealemail.com).

## APPENDIX

[Return to Main Document](#TestSection)

## 

[Figure 1 - Test Case ID: GUI\_01, GUI\_02 B-2](#_Toc463902136)

[Figure 2 - Test Case ID: WAR\_01, WAR\_03, WAR\_04, WAR\_05, WAR\_07 B-2](#_Toc463902137)

[Figure 3 - Test Case ID: WAR\_08 B-3](#_Toc463902138)

[Figure 4 - Test Case ID: WAR\_09 B-3](#_Toc463902139)

[Figure 5 - Test Case ID: CON\_02, CON\_04, CON\_5, CON\_07 B-4](#_Toc463902140)

[Figure 6 - Test Case ID: BJ\_11, BJ\_13, BJ\_15 B-5](#_Toc463902141)

[Figure 7 - Test Case ID: HL\_02, HL\_06 B-6](#_Toc463902142)

[Figure 8 - Test Case ID: T1\_06 B-7](#_Toc463902143)

[Figure 9 - Test Case ID: T1\_07 B-7](#_Toc463902144)

[Figure 10 - Test Case ID: SOL\_04 B-8](#_Toc463902145)

[Figure 11 - Test Case ID: SOL\_05 B-9](#_Toc463902146)

[Figure 12 - Test Case ID: SOL\_07 B-10](#_Toc463902147)

[Figure 13 - Test Case ID: SOL\_08 B-10](#_Toc463902148)



Figure - Test Case ID: GUI\_01, GUI\_02

[Return to UnifiedGUI Test Table](#TestSectionGUI)

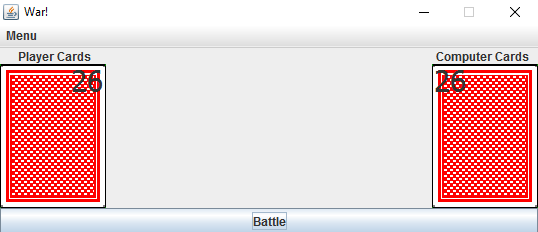


Figure - Test Case ID: WAR\_01, WAR\_03, WAR\_04, WAR\_05, WAR\_07

[Return to War! Test Table](#TestSectionWar)

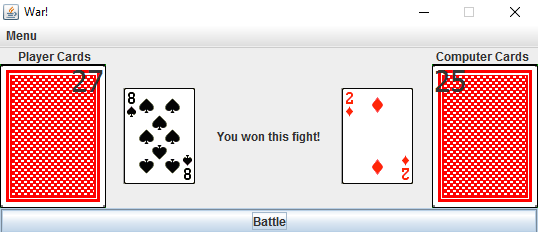


Figure - Test Case ID: WAR\_08

[Return to War! Test Table](#TestSectionWar)

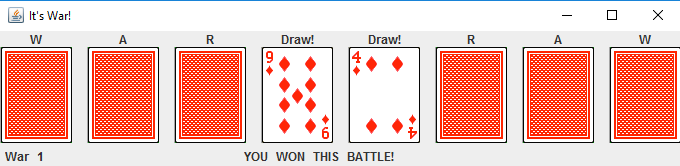


Figure - Test Case ID: WAR\_09

[Return to War! Test Table](#TestSectionWar)

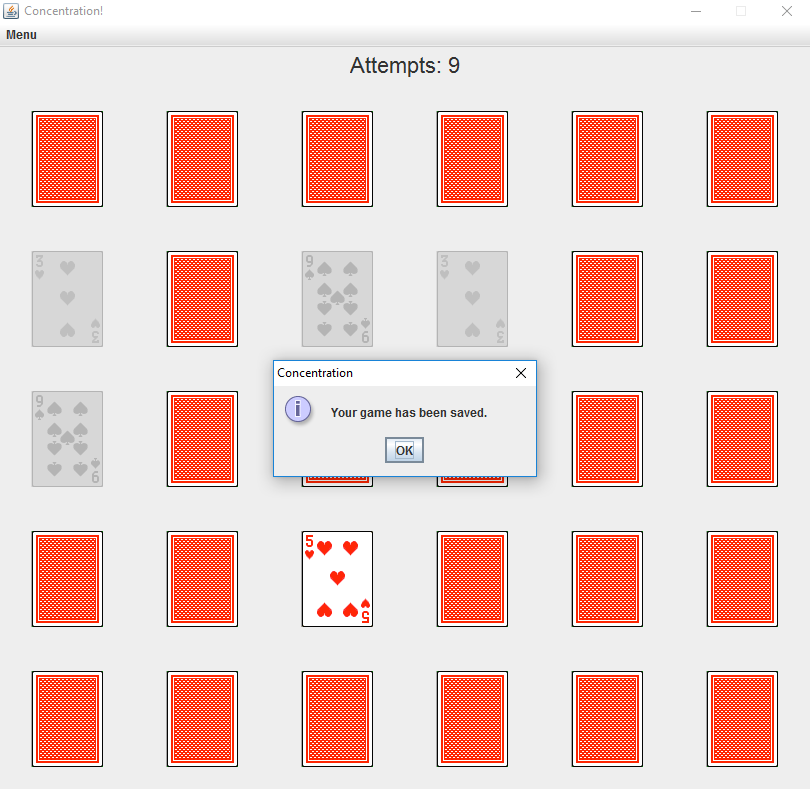


Figure - Test Case ID: CON\_02, CON\_04, CON\_5, CON\_07

[Return to Concentration Test Table](#TestSectionCon)

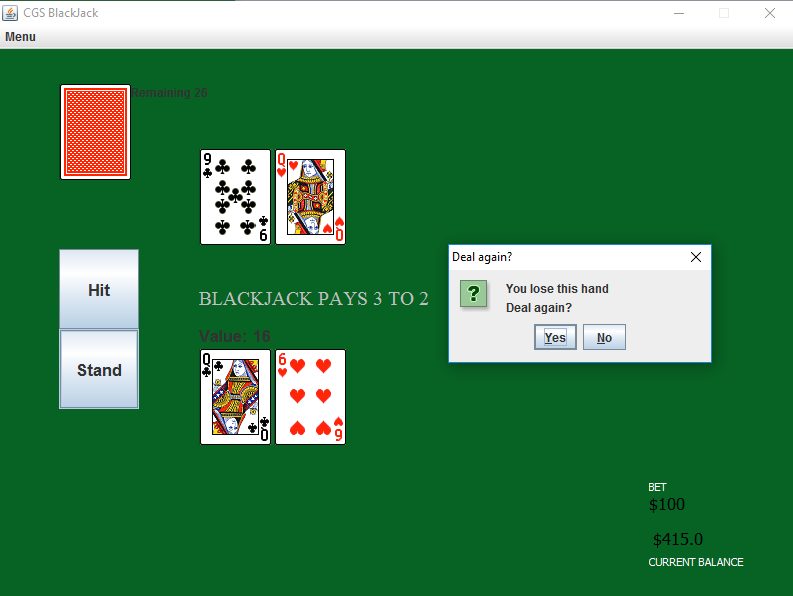


Figure - Test Case ID: BJ\_11, BJ\_13, BJ\_15

[Return to Blackjack Test Table](#TestSectionBJ)

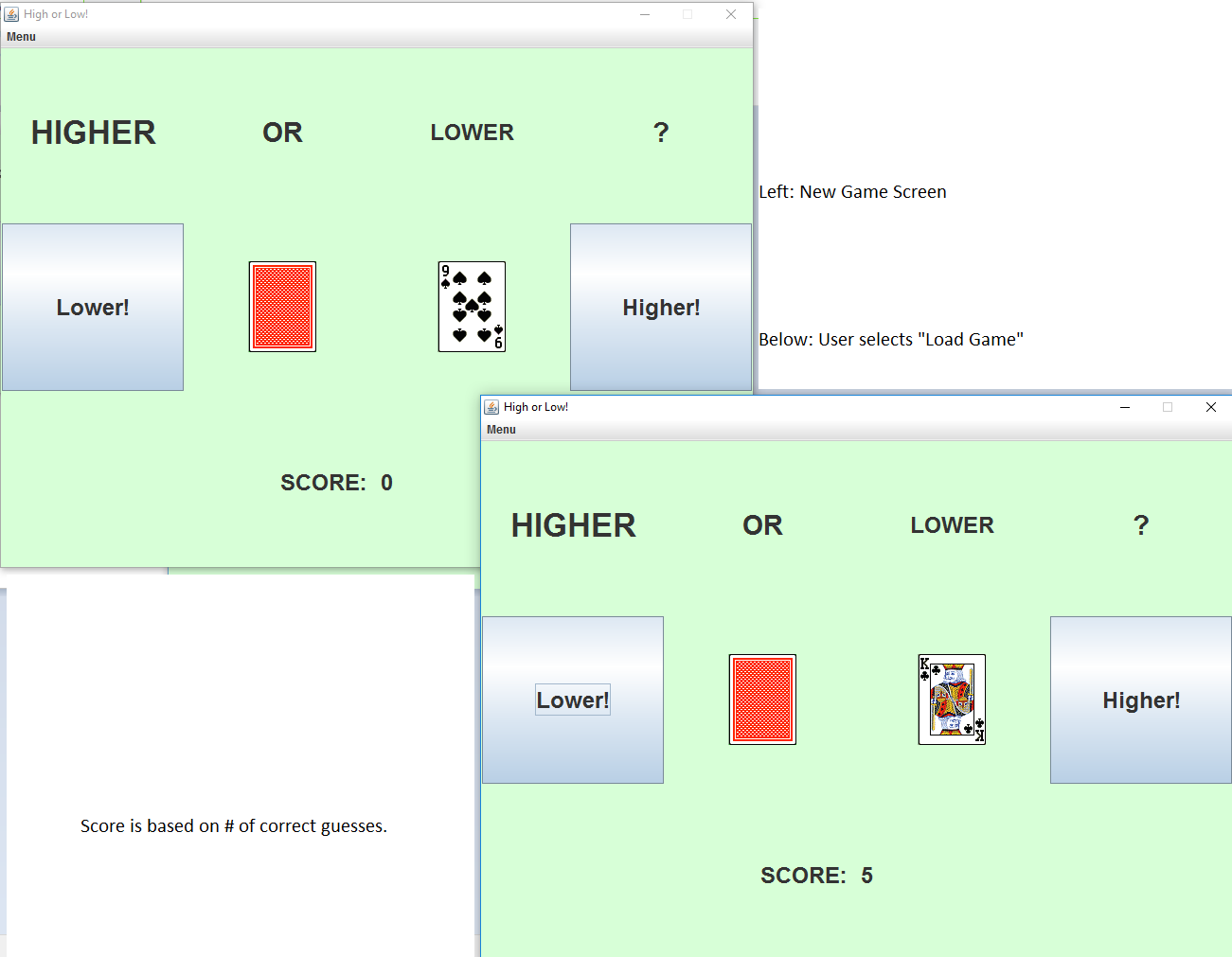


Figure - Test Case ID: HL\_02, HL\_06

[Return to High or Low Test Table](#TestSectionHL)

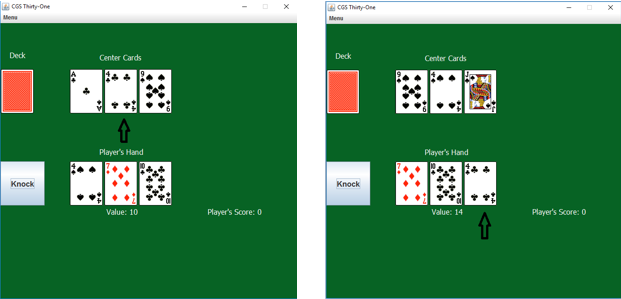


Figure - Test Case ID: T1\_06

[Return to Thirty-One Test Table](#TestSection31)

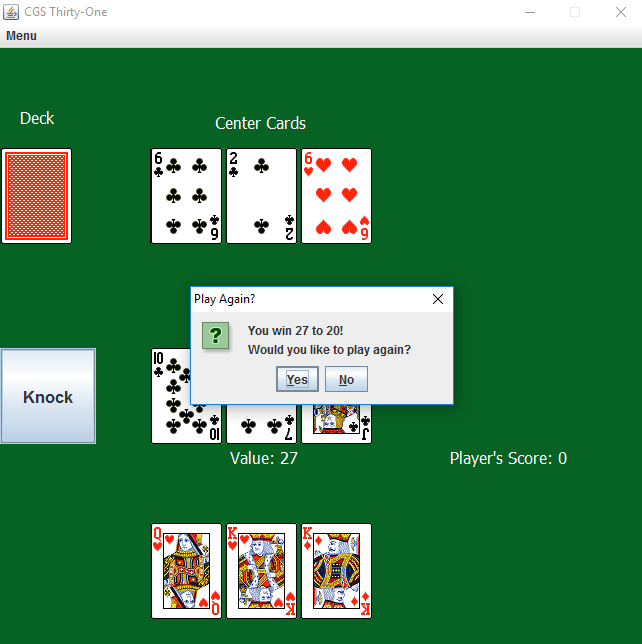


Figure - Test Case ID: T1\_07

[Return to Thirty-One Test Table](#TestSection31)

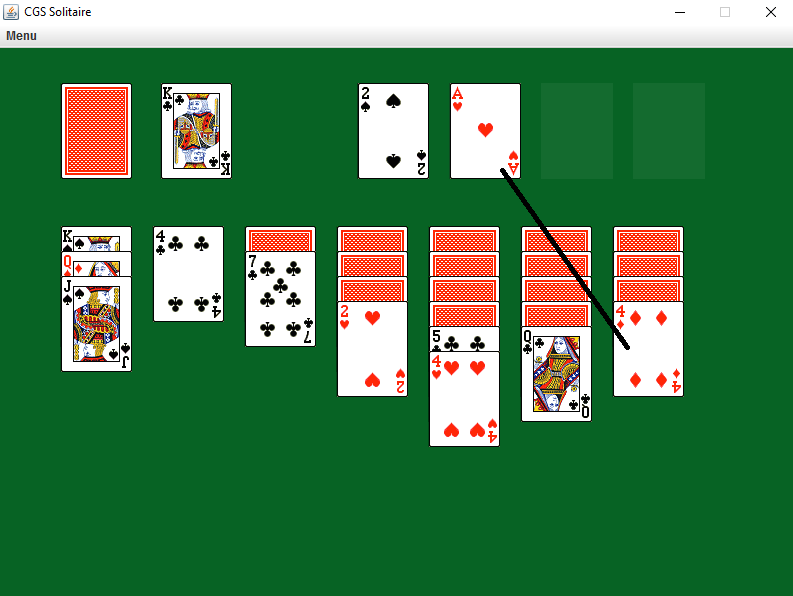


Figure - Test Case ID: SOL\_04

[Return to Solitaire Test Table](#TestSectionSol)

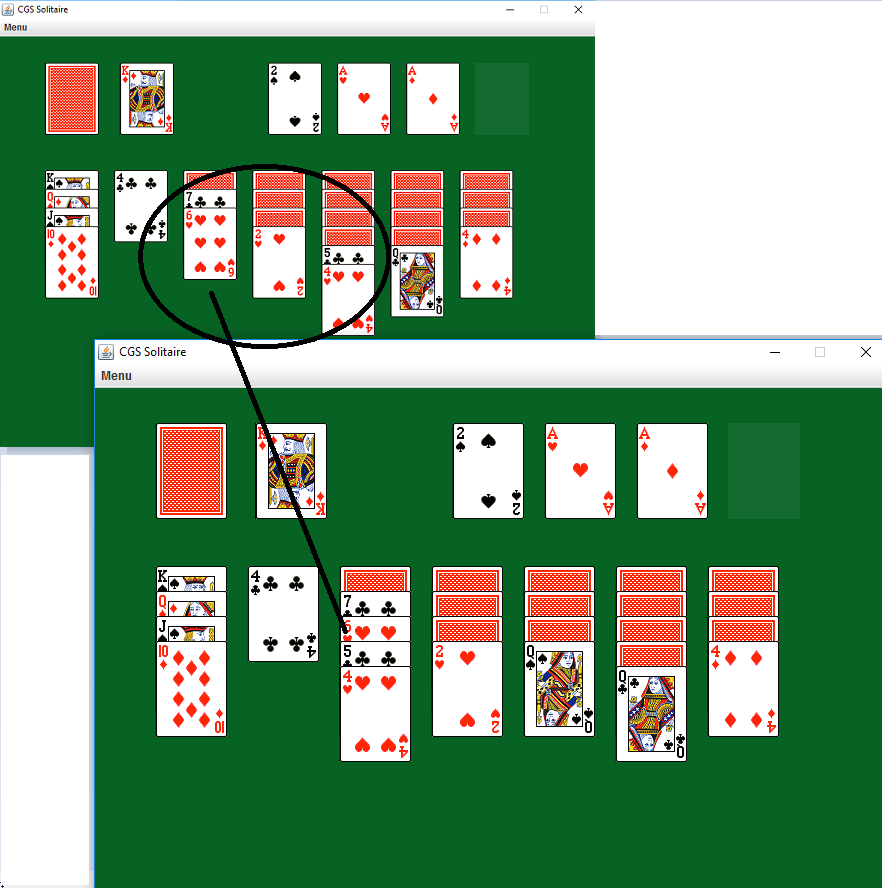


Figure - Test Case ID: SOL\_05

[Return to Solitaire Test Table](#TestSectionSol)

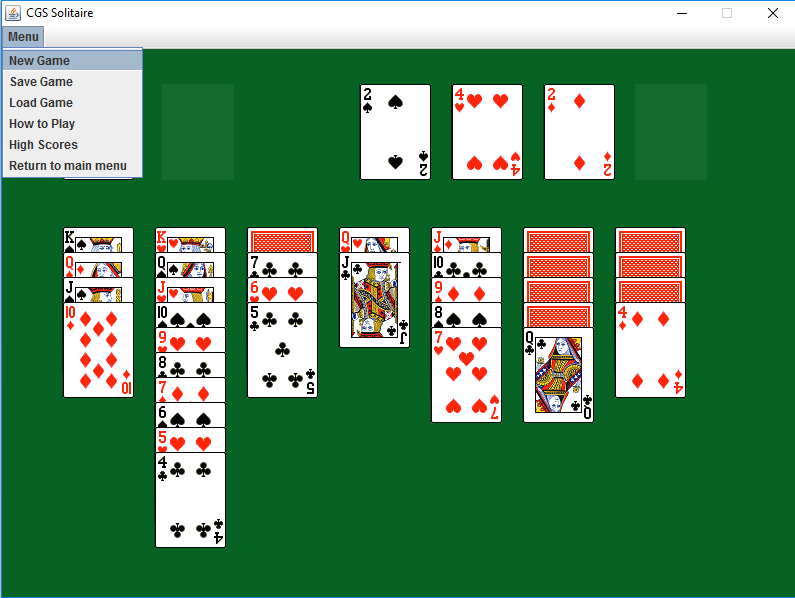


Figure - Test Case ID: SOL\_07

[Return to Solitaire Test Table](#TestSectionSol)

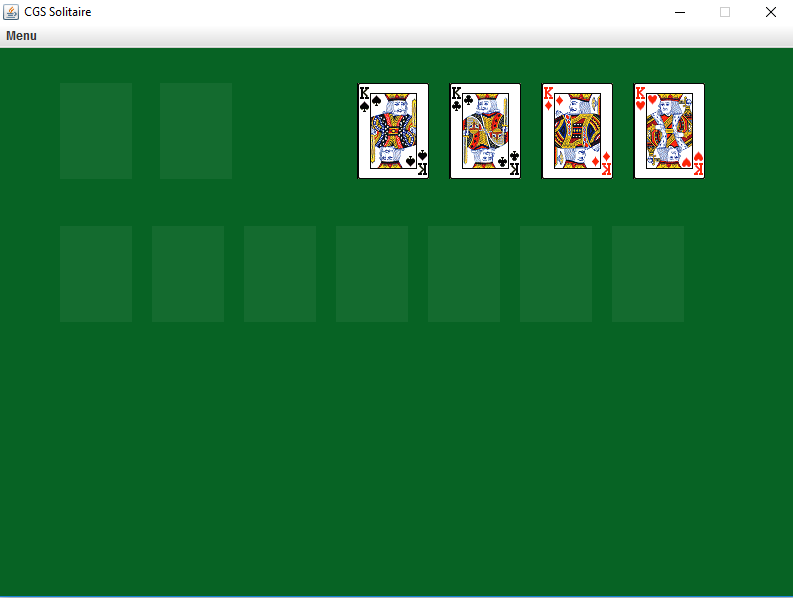


Figure - Test Case ID: SOL\_08

[Return to Solitaire Test Table](#TestSectionSol)