Software Project Final Report

Caravan card game

Alex Studniarz
CSUID: 2824959

Bilal Haider
CSUID: 2726111

Gabriel Surovey
CSUID: 2798110

1.0 Introduction

This software aims to adapt the card mini-game Caravan from Fallout New Vegas, into a standalone desktop application, while making slight improvements to interaction.
The ultimate goal of this software is to allow users to play a full game of Caravan and allow users to customize their Caravan deck for use in game.

1.1 Purpose and Scope

This is an entertainment software that will facilitate the process of playing a card game called Caravan by ensuring the user does not violate any of the rules of the game. Users will interact with the software through a main menu user interface (UI) which will allow them to select menu options. These menu options will either bring up other user interfaces (example: a deck builder UI) or exit the program to desktop.

1.2 Product Overview

What's inside the scope of this project?:

- Bare minimum user interfaces required to play the game: main menu UI, deck building UI, game UI.
- The ability to play a full game of Caravan.
- Basic programmer art.

What's outside the scope?:

- Online multiplayer.

- An interactive tutorial.
- Intricate/bespoke sound design.
- Complex art style.
- Major expansions to the core gameplay loop.

1.3 Terms, Acronyms, and Abbreviations

- **UI**: User Interface
- **Caravan**: A tract where cards can be placed. Each player has 3 Caravans.
- **Value:** The sum of numbered cards within a Caravan.
- **Caravan direction**: Caravans have a direction, either ascending or descending numerically. Direction dictates the types of cards you can place in a Caravan, forcing all subsequent cards to continue the numerical direction set by the second number card placed in the Caravan.
    - **Ascending Caravan**: A Caravan where only cards of increasing value can be placed.
    - **Descending Caravan**: A Caravan where only cards of decreasing value can be placed.
- **Sold**: When the sum of cards in a Caravan is in between the threshold of 21-26, the Caravan is considered to be sold.
    - **Overburdened**: A Caravan whose sum value is below 21.
    - **Underburdened**: A Caravan whose sum value is above 26
- **Outbid**: When two opposing Caravans directly across from each other are both sold, the Caravan that has more value than the other is said to Outbid the other. The player who outbids 2/3 of their opponents Caravans wins the game.
- **Deck**: The pool of cards that you draw from to place in your hand.
- **Hand**: The cards that are able to be played during a turn.
- **Set**: A standard deck of 52 playing cards. Distinguished from deck purely to avoid confusion with Caravan decks

2.0 Project Management Plan

Submissions and deadlines for the project:

- Project plan 9/16/25
- Software requirement specification 9/30/25
- Software design specification 10/14/25
- Demonstration of initial version of software 10/28/25
- Test plan 11/4/25

- Project Deliverables 12/4/25 (This is the 'drop dead' delivery date)

2.1 Project Organization

We used discord to communicate and make plans with each other, github for version control and collaborative development, Google drive to collaborate when writing documentation, and Godot to create the Caravan Desktop program.

2.2 Risk Analysis

- Scope creep - Game development is notorious for scope creep. We are reducing this risk by keeping a strict scope and reminding ourselves of the small amount of time that we have.
- Inaccurate estimations - Inexperience with developing software means our estimations for deliverables may not be accurate. Constant and clear communication with project members can reduce the severity of missing deadlines.
- Low productivity - Low productivity can be caused by internal (as in by the project) and external factors outside of our control. For the factors we have more say in, we can mitigate some of the risk by creating clear tasks for team members to clear off the list.

Name of risk	**Probability**	**Impact**
Scope creep	High	Serious
Estimation	Moderate	Serious
Low Productivity	Moderate	Serious

2.3 Deliverables and schedule

Since we are a very small team we will by employing elements of scrum into our project management. Project specification, design, and implementation will interleave each other in our process. Once or twice a week we will hold a meeting in which we will discuss what needs to be done, what we are currently working on, and any concerns we may have concerning the project and deadlines. Additional to these weekly meetings we will be in constant communication with each other to discuss aspects of the project. The nature of game development is always in flux, thus we will keep plans somewhat loose while keeping up with project deadlines.

September	October	November	December
Project plan due 9/16 Learn the basics of the Godot game engine 9/22 **Software requirement spec due 9/30**	Working version of the main menu UI10/3 Working version of game UI 10/10 **Software design specification due 10/14** The ability to place cards 10/26 **Demonstration of initial version of software 10/28**	**Test plan due 11/4/25** The ability create a deck 11/10 Deck creation UI by 11/17 CPU opponent by 11/24 Remaining time is spent on polish	**Project Deliverables (drop dead deadline) 12/4**

3.0 Requirement Specifications

3.1 Stakeholders for the system

- Group members
- Professor and TA
- Anyone with an interest in playing the game

3.2 Use cases

The only use case we see for our software is playing the game itself for recreation/entertainment purposes.

3.3 Non-functional requirements

Non-functional requirements:

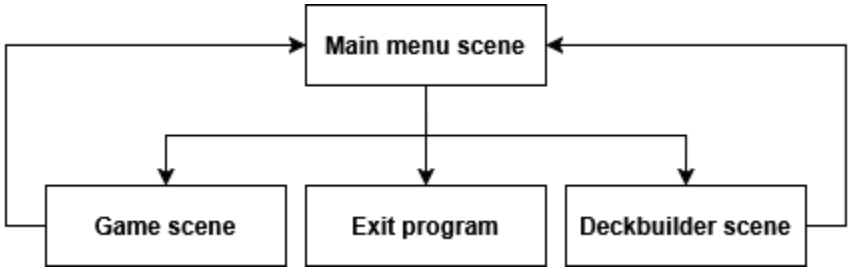
- The open source game engine Godot, shall be used to create the software
- The game must work on the Windows operating system
- The software shall be able to be operated with only the left mouse button

4.0 Architecture

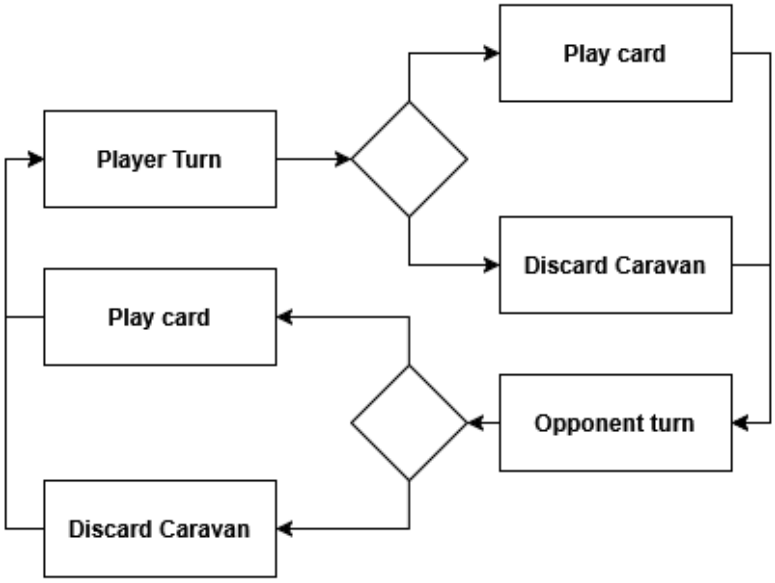
Going in we had no particular architectural pattern in mind. We concepted an architectural design for the main game, but this was not based on any pattern. Eventually the structure of the code began to deteriorate as a consequence of our architecture not being very well thought out.

4.0.1 Highest level architectural diagram

This is what the highest level view of the current architecture of our program looks like.



4.0.2 Game scene turn system diagram

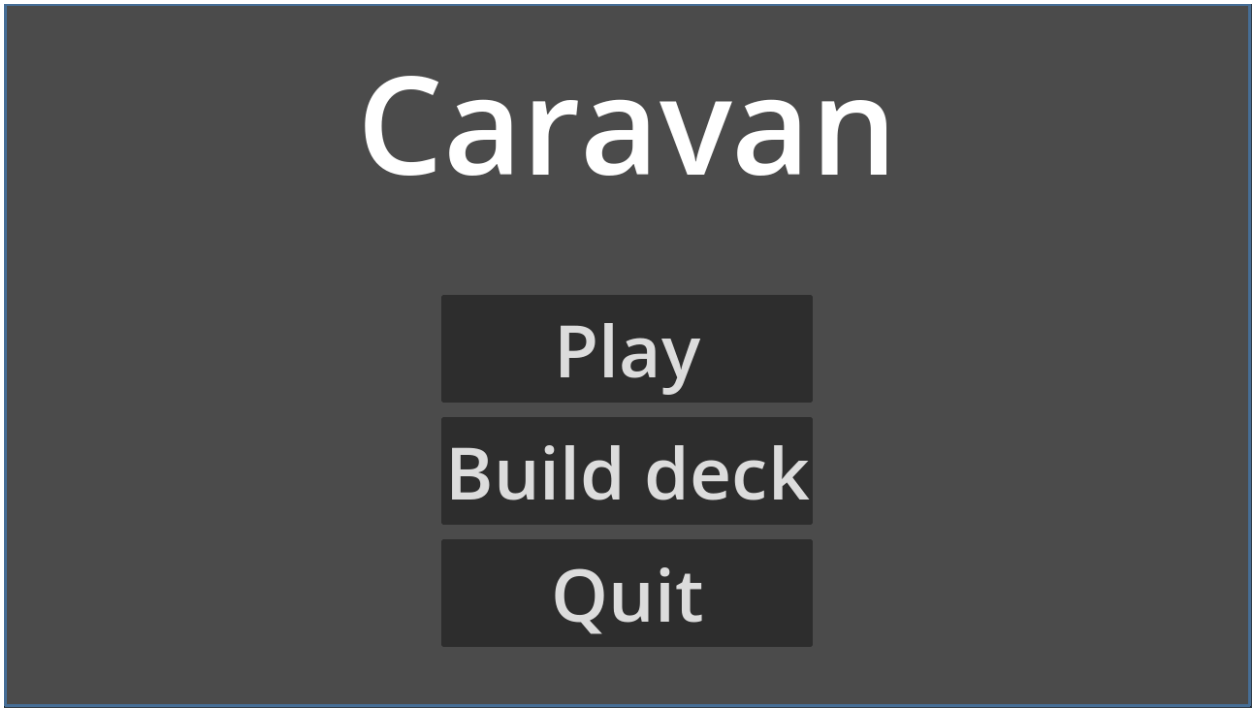


5.0 Design

5.1 User Interface design

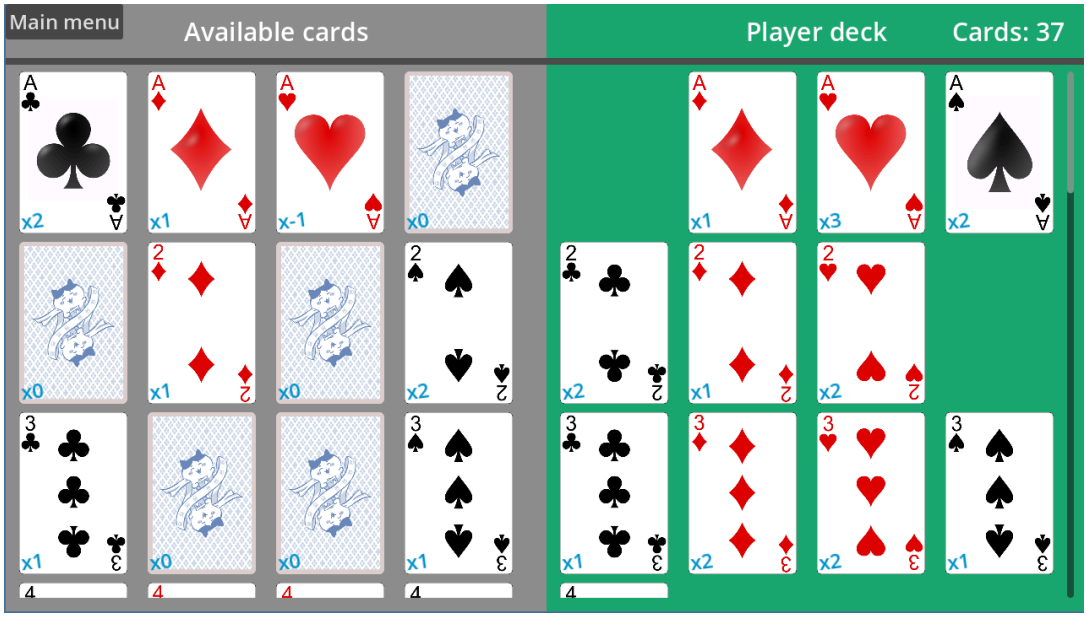
5.1.1 Main Menu

There isn't much that can be said about our main menu interface design. It is very simple and gets the job of transitioning to other scenes in the project done.



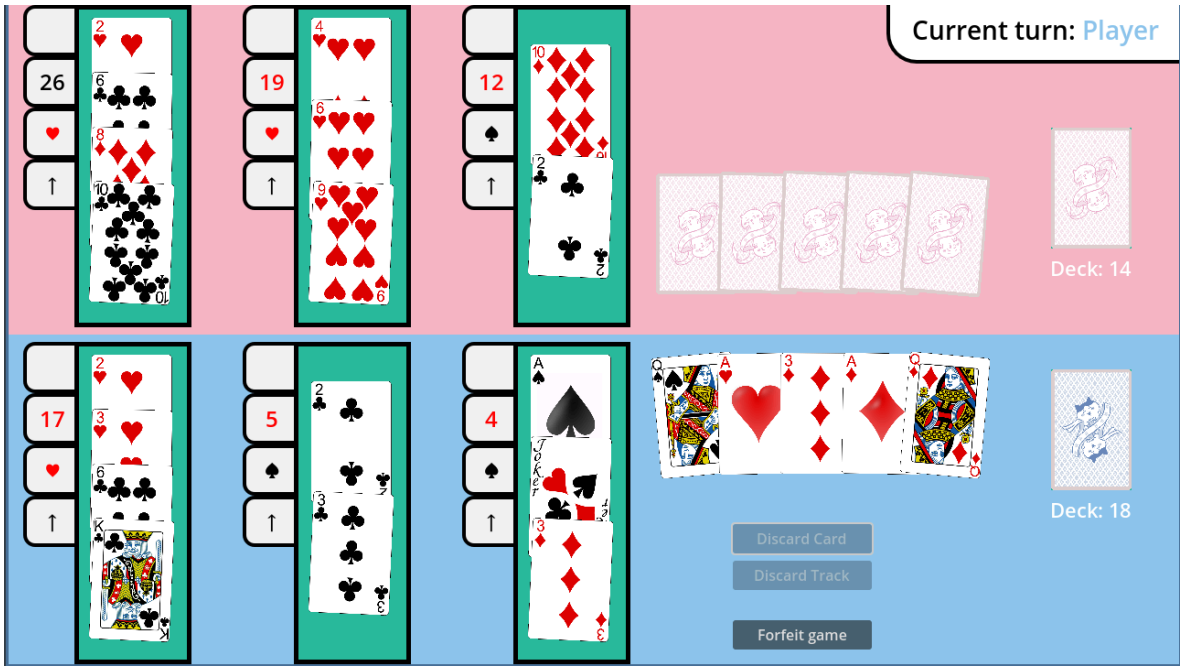
5.1.2 Deck Builder

We went through a few iterations before we found one we liked. The left side is the cards that are available to be chosen, along with how many are allowed. You can choose a card by clicking on it. This will move the card to the right side of the screen and add one to the card counter display in the top right. The player needs at least 30 cards in their deck to be able to click the main menu button on the top left to save their deck and return to the main menu.



5.1.3 Main Game

We gave the main game screen a new look too. Now the Caravans display the direction, suit, and indicate if one Caravan is outbidding another. The value indicator on the Caravan turns red when the Caravan is under or overburdened to indicate it is not “sold”, and turns black when it is in the value range of 21-26 to indicate that it is sold. We also added highlights to cards and caravans to indicate that they are selected and project card placements to show the player they are able to place a card.



6.0 Test Management

These test cases were planned but never actually implemented into our project's unit tests. Our unfamiliarity with test driven design made us stray away from that strategy and by then we ran out of time to really add them to the project.

6.1 A complete list of system test cases

Main menu test cases

ID	test_play_button
Test input	Simulated “Play” button input
Expected output	Game scene loads successfully

Description	Tests that the “Play” button transitions from the main menu to the game UI without error.

ID	test_build_deck_button
Test input	Simulated “Build deck” button input
Expected output	Deck builder scene loads successfully
Description	Tests that the “Build deck” button transitions from the main menu to the deck builder UI without error

ID	test_quit_button
Test input	User clicks Quit on Main Menu
Expected output	Main menu scene exits correctly
Description	Ensures the program terminates cleanly when “Quit” is selected

Deck builder test cases

ID	test_deck_card_requirements_valid
Test input	Simulated main menu button input with a card count of ≥ 30 cards
Expected output	Scene changes to main menu
Description	Confirms UI updates and enables saving only once the deck meets the minimum card count

Main game test cases

ID	test_caravan_suit
Test input	Card object “ace_of_spades” is passed to caravan
Expected output	caravan_suit = “spades”
Description	Tests correct initialization of Caravan suit

ID	test_caravan_direction
Test input	Card objects “ace_of_spades” and “2_of_clubs” are sequentially passed to caravan
Expected output	caravan_direction = “ascending”
Description	Ensures direction is determined properly after second card placement

ID	test_caravan_removal
Test input	“Ace_of_spades” and “2_of_clubs” are passed to caravan, “remove_caravan_tract()” method is executed
Expected output	caravan_sum = 0 and caravan_direction = “none” and caravan_suit = “none”
Description	Tests that the remove caravan tract method is correctly working

ID	test_king
Test input	“Ace_of_spades”, “2_of_clubs” passed to caravan. “King_of_clubs” is placed in position ahead of “2_of_clubs”
Expected output	caravan_sum = 5
Description	Tests that kings are working as intended

ID	test_queen_direction
Test input	“Ace_of_hearts”, “10_of_diamonds” passed to caravan. “Queen_of_diamonds” is placed in position ahead of “10_of_diamonds”
Expected output	caravan_direction = “decreasing”
Description	Tests to confirm that queens reverse the direction of the caravan

ID	test_queen_suit
Test input	“Ace_of_hearts”, “10_of_diamonds” passed to caravan. “Queen_of_clubs” is placed in position ahead of “10_of_diamonds”
Expected output	caravan_suit = “clubs”
Description	Tests to confirm that queens change the suit of the caravan to the suit of the queen

ID	test_jack
Test input	“Ace_of_hearts”, “10_of_diamonds”, “king_of_diamonds”, “king_of_hearts” passed to the caravan. “jack_of_diamonds” is placed in position ahead of “10_of_diamonds”
Expected output	caravan_sum = 1
Description	Tests to ensure that Jacks properly remove the card they were played on and remove any face cards that are “attached” to the number card

ID	test_joker_ace
Test input	“Ace_of_clubs”, “5_of_spades”, “king_of_clubs”, “10_of_clubs” passed to player and opposing caravans. “joker” is placed in player caravan a position ahead of “ace_of_clubs”
Expected output	Player caravan_sum = 11 and opposing caravan_sum = 10
Description	Tests to ensure that a Joker played on an ace will remove all cards of that ace’s suit, except for face cards and the ace it was played on, from the whole table.

ID	test_joker_number_card
Test input	“7_of_hearts”, “7_of_clubs”, “7_of_diamonds”, “7_of_spades” passed to player and opposing caravans, “joker” is placed in position ahead

	of “7_of_hearts” in players caravan
Expected output	Player caravan_sum = 7 and opposing caravan_sum = 0
Description	Tests to ensure that when a Joker is played on a number card, all cards of that number card regardless of suit, with the exception of the card it was placed on top of are removed from the whole table

ID	test_win_condition
Test input	Player side has two caravan tracts set with sums = 26, adjacent opposing caravans have sums = 25
Expected output	“Player wins”
Description	Tests the win condition for when the player outbids two of the opponents caravans

ID	test_loss_condition_empty_deck
Test input	“player_deck” initialized with 1 card, “draw_card()” function is executed
Expected output	“Player looses”
Description	Tests to see if the loss condition is properly triggered when a deck runs out of cards.

ID	test_different_suit_same_number
Test input	Player plays “4_of_hearts” on “4_of_spades”
Expected output	“4_of_hearts” is passed to caravan
Description	Tracts are allowed to have same value card form a different suit to be placed on them, if the card is a different value then it is not allowed

ID	test_caravan_sold_status_overburdened

Test input	caravan_sum is set to 27
Expected output	caravan_sold = “false”
Description	Tests to ensure that the logic to determine the sold status of a caravan is functioning. If the caravan has more than 26 cards its status should not be sold.

ID	test_caravan_sold_status_underburdened
Test input	Caravan sum is set to 20
Expected output	caravan_sold = “false”
Description	Tests to ensure that the logic to determine the sold status of a caravan is functioning. If the caravan has less than 26 cards its status should not be sold.

ID	test_caravan_sold_status_lower_bound
Test input	Caravan sum set to 21
Expected output	caravan_sold = “true”
Description	Tests to ensure that the logic to determine the sold status of a caravan is functioning. If the caravan's sum is within the range of 21-26 its status should be set to sold.

ID	test_caravan_sold_status_upper_bound
Test input	Caravan sum set to 26
Expected output	caravan_sold = “true”
Description	Tests to ensure that the logic to determine the sold status of a caravan is functioning. If the caravan's sum is within the range of 21-26 its status should be set to sold.

ID	test_caravan_card_placement
Test input	Card objects “5_of_clubs” and “6_of_spades”

	are passed to the caravan. Card object “2_of_hearts” is passed to the caravan.
Expected output	“Invalid card placement”
Description	Test to ensures invalid placement card placement fails correctly

6.2 Techniques used for test case generation

The main technique used observation of the original card mini-game from Fallout: New Vegas. Additional test cases were generated from interviews with the on-site customer (one of our group mates was designated to be the on-site customer).

7.0 Conclusions

7.1 Outcomes of the project

The main goal of this project was to be able to play a full game of Caravan from start to finish and to be able to customize your deck for use in-game. These goals were achieved, the game is fully playable and the deck building (we feel) is better than the original game. Despite the messy development and architecture degradation, overall we would call this project a success.

7.2 Lessons learned

The importance of having a plan for the architecture of your code and sticking to that planned architecture. Because we didn’t have a knowledge of architectural design and patterns when starting the project, the code quickly degraded which made it very hard to make changes in the final stages of the project. Starting the project with a specific architectural pattern in mind would have benefited us greatly, making software evolution much easier. The importance of keeping a very tight scope to avoid scope creep was hammered home. If we didn’t keep to our tight scope, our project would have surely failed. We also learned the importance of keeping our nostrils open for “bad smells” in our code. There were several instances of bad smells in this project. One “bad smell” that appeared frequently was duplicate code. Very similar code was copied and pasted for the sake of convenience at the time and that re-used code was never generalized. If these sections of duplicate code were turned into functions that would have greatly helped readability and maintainability. Another “bad smell” that frequently arose was

long methods. These long methods further harmed the readability of the code and could have been re-designed to be much shorter.

7.3 Future development

- Refactoring the code to be cleaner and more understandable
- Give the CPU the ability to play face cards against the player
- Re-do the collision code entirely