Joseph Allen Manual for CPP

[jallen6@ramapo.edu](mailto:jallen6@ramapo.edu)

# Bug Report:

* Ai fails to find any sets from the Player::findSelectableSets functions for the purposed of capturing
* When loading from a file with an empty deck, occasionally there are the following issues:
  + The round doesn’t properly end until a player attempts to play card. This is a non-fatal error that the program handles
  + When round ends from a save file, there is at least one (unknown) control path which does not reset the loading flag
    - This causes the Players to load in for a new game, and the Deck and table to load from the file
* When a save file is loaded, there is a control path which doesn’t initialize the topOfDeck iterator to the correct portion of the deck
  + This causes no cards to be dealt to the players, and a potential unhandled exception
* Extending a build can occasionally attempt to format itself as a multi build, which is not valid. However, this error is after the point of no return and the operation cannot be aborted without causing a card to be lost.
  + The current method of handling is to allow the build to be formatted, and output the value it expects, which matches the value to create it. Ie, if a two was added to extend the build, the entire build will be valued at one
* When the AI prints out moves or recommendations, builds are calling the wrong toString function
  + They call their parent. So, while the important information is being sent, the formatting is not intended.
* Human is able to trail while owning a build

# Feature Report:

## Non-implemented:

* AI being able to capture sets
  + Logic is there, however the AI is unable to find the cards
* Human and AI being able to properly interact with multi builds
  + Only partial functionality that is buggy
* Extending builds is equally as buggy for both

## Extra:

* Human is able to set their name and the computer has a custom name
* Program detects all required cards for the human and automatically selects them
* If the human only has one card left in their hand, it will automatically select it as the played card.

Description of Data Structures and classes

**Build:** This class inherits from the Card Superclass and is responsible for maintain a vector of Card\*, which are all the cards in the build

**Card:** Superclass for Build and PlayingCard class, and used for polymorphic behavior. Keeps tracks of the suit, symbol, numeric value, and functions for checking equality between the objects.

**Casino:** Main file which contains the Main for the program, and instantiates the tournament class

**Client**: A static class which manages all the output and input from the console. It handles the multiple types of receiving inputs and making sure its in a valid range

**Computer:** Inherits from the player class, which and formats the PlayerMove Struct to be returned for the move to be processed by the table class

**Deck:** Manages the vector and iterator to handling the cards to be dealt, and tracking when all cards have been dealt

**Hand**: Manages the cards in a hand, including adding and removing the Cards from its member vector

**Human:** Inherits from the player class and instantiates helper functions to fill out the virtual doMove function. Relies on getting input from the user to decide what cards to choose, and uses the Player Class functions to validate

**Player:** Pure virtual superclass to the Human and Computer Class, which guarantees the implementation of the do Turn function and the validators

**PlayingCard:** Inherits from the Card class and used to represent a single card that is in no way a build.

**Round:** This class is responsible for spawning the table and having it run its cycles until completion, then extract the data so the tournament can reach it.

**Serializer**: This static class is responsible for reading in data from a save file, saving data in the correct format, and providing helper functions to manipulate loaded data, such as remove headers and tokenize on spaces

**Table:** This class is the main driver. This class creates and manages the two players, the deck, and the loose cards on the table. Once a player make a move this class is responsible for moving the cards to the correct place, both the loose cards or the pile of the appropriate player.

**Tournament:** This class is the entry point into the game, and checks at the end of every round they have been completed, if not spawn a new round. If it have, find who wins and outputs.

**PlayerMove** Data structure: This struct contains the move a player is going to make, such as the enum for the move type(Player::Actions), the played card, and the indicies of all the selected cards.

# Log:

Sep 7, 2018:

Design:

-Spent time playing the card game to learn the rules (1.5 hours)

Total 1.5 Hours

Writing:

-Created the required classes .cpp and .h files(Card, Computer, Deck, Human, Player, Round, Tournanment) and Casino.cpp which contains the main() (.5 hours)

-Added basic Round loop, which creates the tornament objects, which then starts a round object. That round object will flip a coin, intialize counter vars, then wait for the round to complete (not implemented) (1 hour)

-Implemented logic to end the torunament when one or both players reach 21 points, and outputs who won (.1 hours)

-Implemented Card Class skeleton so a card can keep track of its suit and symbol (.1 hours)

-Implemented the skeleton of the Deck class, which is resposnbible for creating all 52 cards, storing them in a vector. Once there, it has the ability to shuffle the deck, and draw the top card. (1 hour)

-Edited output logic for coin flip to be more clear (.1 hour)

-Designed and implemented a Table class to handle the per cycle (Each player playing a card) logic which is managed by the Round class. This implementation kept track of both hands, piles, and the loose cards on table. (1.5 hours)

Total: 3.8 hours

Day Total: 5.2 hours

Sep 8, 2018:

-Made the Player class pure virtual and allowed for a Human object to ask and set its name in the overloaded setName() function. The computer class overloads the same function and sets to the same string(.5 hours)

-Editied the Card's class toString() function to remove garbage data that occasionally got picked up (.1 hours)

-Implemented the board's printBoard() function. This function outputs both player's name, their hands, and the cards on the table (.5 hours)

-Editied logic for coinflip in the Round class by forcing to lowercase, and therefore simplifying the if, else checks. In addition, the program now flushes STDIN to improve handling of bad user input (.5 hours)

-Redesigned the hierarchy for the Player class inheritence.

Former method was to have each child class respond with overloaded function to a non-virtual public parent function.

New design is to have that function be pure virtual, and each child having full control over the implmentation, and then having non-virtual validator functions to ensure both abide by the same rules.

(.5 hours for this section)

-Edited code in Table and Human for better readbility and useability:

In Table, the player hands are stored in an array instead of seperate variables to allow for looping instead of hardcoding each one

In human, the Action enums (Caputre, Build,Trail) were moved to the parent class so the AI class can use them when implemented.

(.5 hours for this section)

Day total: 2.6 hours

Sep 9, 2018:

-Implemented the ability to capture cards of the same symbol:

Changed the doTurn signatre (in Player) to to vector<Card> instead of a single int, this is to allow for the Player logic to see the cards

Card class has a function [checkCapture(Card)] to check if a target card can capture it

Human class doTurn overload recieved a skeleton implementation:

Prompts the user for which action they want to take, (Capture, Build, or trail)

Ask what card in their hand they want to play.

Ask what card they want to select on the table.

Then use a switch statement to handle the three action cases

Above took 2.5 hours

-Editied code through program to fix compiler warnings for intened casts that may cause issues:

Such as using a signed int i to loop through a vector. This got changed to an unsigned int

Explictly casted the srand seed to unsigned int

Above took .2 hours

-Implemented function symbolToNumericValue() in the card class so cards can convert the face symbol to numeric representation. (.1 hours)

-Edited all getters through the program to be const to confirm to coding standards (.1 hours)

- Implemented abilty to select multiple cards on the table

Card class got another checkCapture overload that accepts a vector of Cards

Player class implemented a new non-virtual function to validate selections from both child classes

Human changed it gets input functions to receive a vector of ints instead of a single int to handle this

Above took .5 hours

- Edited Human's promptForCardToUse(int, bool) to keep track of previous inputs and prevent the user from inputting the same card .5 hours

-Testing to ensure that capturing cards works as intended for Human:

\* Capturing identical Card: Works

\* Capturing all identical Cards: Not working

\* Capturing a set: Works

\* Denying a capture on an incorrect set: Works

\* Error handling on a user not inputing a valid action: Works

\* Error handling on a user not inputing a valid number: Works

Above took 1 hour

Day total: 4.9 hours

Sep 10th, 2018:

- Created build class, and have it add cards to the build with no logic checking .2 hours

- Trailing implemented with no error checking .1 hours

- Debugged an error with indices not lining up with the displayed indices

This was caused by an errornious call in Human which pre emptively removed the card before all input was taken.

Above took .4 hours (.3 hours to identify and .1 hours to fix)

Day total .7 hours

Sep 11th, 2018:

- Edited Table class to have the player piles in a parrell array to the hand array .2 hours

- Moved hands and pile from the table class to the player class after speaking to you in your office.

All logic for adding and removing from hand was removed from the table class

FillHand() in table now passes the drawn card to the appropriate player object to handle storing the card

Implemented a default constructor to the card class that sets it to an intialized error value if called

Implemented a toFormattedString function to the Player class which returns the cards in the hand with their index value

Converted the caputre and trail in Player to use the new member variables

Above took 2.5 hours

-(Resolved a merge conflict between the version on my laptop and desktop. This reminded me to always sync with the repo before programming .2 hours

- Converted all output in all classes to use a static class (Client) to handle outputs and inputs.

For outputs, client provides and outputString function, which outputs a string to cout, and a outputError function which outputs a string to cerr.

Client also provides ability to get input of types: char, string, and ints.

The program can also apply these constraints to the inputs:

Chars: Have to be on the valid selection list, and can force to lowercase

ints: Have to be on a continous range.

Above took 2 hours

-Edited all mutators to now return a bool instead of void to abide by coding standards .3 hours

Day total: 5.2 hours

Sep 12th, 2018:

Edited Build::addCardToBuild to ensure the card is valid (does not exceed 14) .1 hour

(Resolved another merge conflicts between computers, and this time learning my lesson) .5 hours

-Implemented builds to work in ideal conditions only, where the Player has the correct card in hand, and the card they played sums to the target value (.2 hours)

-Failed to implement merging of the builds into the normal card vector with polymorphism. At the end of this endevour I incorrectly identifed the issue was with inheritence. The actual issue was with object slicing in the containing data structure. 1 hour

-Added new class PlayingCard which inheriets from Card. This was after speaking with you and you saying this would be a better implementation. This class will eventually take the role of Card, and Card will become pure virtual. (.2 hours)

Day Total 1.9 hours

Sep 13th, 2018

-Fixed an error with the Build's class toString function not displaying the formatting correctly .1 hour

-Created a hand class, and moved all the card vectors into the class, with accessors to handle the data. .5 hours

- When a player creates a build, that card becomes reserved in their hand, which the Player's hand manages, and will prevent it from being used in a trail .5 hours

-Spilt the fuction code Player::Check trail into two functions

checkResereved which checks if the provided card is resereved

and checkTrail now returns the opposite of what the helper function returns

This allows for the code to be reused in other functions which need to know if the card is resereved

Above took .2 hours

-Edited builds to be identifed by the player name string instead of by an index

This is due to every class having seperate indexing schemes which are not guareenteed to always be identical

Above took .2 hours

Day Toal 1 hour

Sep 14th, 2018

-Debugged why builds toString function was not being correctly used (per Sep 12ths log) was caused by object slicing on the parent object. After speaking with you in your office, implemented your solution of moving the owner information to the parent class, so it is visible, and the owner for the cards is who played them

Took 1 hour total. .7 hours to debug the issue, .3 to implement solution

Implemented multi builds in an untested state

The reason they are untested is there is no proper way to test them until serialization is implemented.

However, as they use the same class, and it is two builds instead of two cards, the overrall idea will work, however the edge cases may be office

Above took .5 hours

-Editied the table class to respect the coin flip in the round class .1 hour

-Removed commented out code in the Table class which was left over from the move to the Player classes (Sep 11th log) .1 hour

Day total 1.6 hours

Sep 16th, 2018

-Created a yes/no input function wrapper inside of Client to clean up code that prompts only for that. Which keeps calling code clean (.1 hours)

-Implemented a skeleton of Seriliazation, which includes opening the file, and being able to tokenize each line

\*New class Serializer which handles the opening, parsing, and storing of loaded in data.

\* Tournament class is responsible for calling the init function of the static class in its constuctor

\* Tournment checks for save loading before the first round

Above took 2.5 hours

Day Total 2.6 Hours

Sep 17th, 2018

-Failed implementation of loading in the data from a file and using a map and enums to parse the data through a switch statement. All code discarded due to failing at parsing in any efficent method(2 hour)

-Created the Serializer::PlayerInfo struct and all its getters and mutators which allow for data to be read by the Player classes (.5 hours)

-Implemented a Serializer::parseLine function which can tokenize a given string and return it as a vector of strings (.5 hours)

-Implemented functions to read N lines from a fuction and have them returned as a vector of strings, where this function skips over blank lines. (.5 hours)

-Overloaded constructors for Round, Table, and Human to be able to load in save data for the human class (1 hour)

-Improved upon the existing function to load in the save file to access the new helper functions that were created above (1.5 hours)

Day Total: 6 hours

Sep 18th, 2018

Updated Rubric for submission today (.5 hours)

Commented Classes (3 hour)

Day total 3.5 Hour

Sep 19th, 2018:

Modifed Deck::intializeCards() to include the ability to check to see if there are any decks loaded through the Serializer and restore the state (1 hour)

Modifed Table::fillLooseCards() to be able to intialize all table cards without formatting them into the builds (1 hours)

Extended the addition to Table::fillLooseCards() to properly load in builds and multi builds (1.5 hours)

Day Total 3.5 hours

Sep 20th, 2018:

Changed the build class to use pointers to store internal member data to prevent accidental recursion and object slicing (.5 hours)

Optimized Table::fillLooseCards() to remove a loop and restructured redundant code to be called only once (1 hour)

Day Total: 1.5 hours

Sep 21st, 2018:

Fixed Tournament::checkForSaveFileLoad() to properly respond to user input instead of always assuming yes (.5 hours)

Debugged an issue when not loading a save file that was causing a segmentation fault. Caused by an arronous >= (.5 hours to debug and fix)

Fixed Build Copy Constructor to be more efficent and properly apply the suit which allows for overloading and properly output when loaded from a save file (1 hour)

Added Table::actionMenu() to display the required menu and respond to the inputs (.5 hours)

Day Total: 2.5 hours

Sep 22nd, 2018:

Serializer::createSaveFile() to write data from the variables properly to a save file, minus for build owner. (1.5 hour)

Implemented Deck::toString() which allows the deck to be saved (.1 hour)

Implemented Hand::toString() which allows the hand to be saved (.1 hour)

Implemented Player::saveSelf() so the player class can format itself to be saved (.1 hour)

Implemented Table::serilizeAllObjects() which call the save functions for all its managed objects, and send the data to the serializer (.1 hours)

Debugged complier errors which were due to a linking error caused by a missing forward decleration in Serializer.cpp(.2 hours)

Converted Serializer::createSaveFile() to use output streams instead of building a string than writing that string for better efficency (.2 hours)

Tournament now ensures that the score data in the serializer object is up to date any time the score changes (.2 hours)

Rewrote the player class to detect when the player is required to capture a matching face card instead of relying on user input, this is in preperation for A.I implementation (1 hour)

Updated the set caputre logic to work with the new implementation of finding required caputres. This made the new function Human::getOptionialInput which asks for all the sets, and ensures they aren't choosen (1.5 hours)

Created a caputre test save file (.1 hour)

Day Total 5 hours

Sep 23rd, 2018

Updated the calls in the build processing to work with the new changes (.5 hours)

Day total .5 hours

Sep 24th, 2018

Failed implementation of created an algorithm to find caputrable subsets with reasonable accuracy and efficency, which was planned to be used for convience for the Human and logic for the A.I (2 hours)

Some of this may be salvaged, this will depend on the following work

Day total: 2 hours

Sep 25th, 2018

Implemented a first version of Player::findSelectableSets which was salvaged from the work on sep 24th. Function still needs to be debugged for edge cases, however it works in ideal. This is the basis for the A.I classes. (1.5 hours)

Created framework for AI to make its card choices (.5 hours)

Debugged the AI checking for possible builds it can make. Currently it finds valid builds, but also tends to get a few invalid options which cause it to fail (1.5 hours)

Day total: 3.5 hours

Sep 26th, 2018:

Fixed whitespace formatting in Serializer::createSaveFile() to match the provided file, and in addition created a test file to work with the ai (.5 hours)

Allowed the Computer class ot load in its serialization file, in the same manner as Human (.5 hours)

AI has the ability to caputre cards if there is one with a matchig symbol (.5 hours)

Ai now respects resereved card values when factoring in captures (.1 hours)

Changed AI behavior on builds from: Trying the first it finds to Trying all it finds (.1 hours)

Day Total: 1.7 hours

Sep 27th, 2018:

Rewrote the Table logic from using hardcoded checks on the player order to using a queue, which reduced code complexity and allowing saves to resume in the middle of a cycle. (.5 hour)

Debugged issues with moves not being properly formatted when they reached the player class, and deduced it was do to a missed sort and reverse on a branch. To reduce redudant code, moved to being done after it returns as all branches have to do it (.5 hour)

Can now save the build owners, and load them in to the builds correctly (1 hour)

Added ability for Decks to load in a prebuilt deck seperately if no save file is loaded (.5 hour)

Day Total 2.5 hours

Sep 28th, 2018:

Rewrote a large portion of build code for checking for values to allow for builds to properly summed when loading in from a save file (1 hour)

Deck, and piles print out with the table on every loop to meet rubric requirements (.5 hours)

Implemented the AI help functionality, which spawns a Computer with the Human's information, and run its algorithm to get its move. (1 hour)

Fixed bug where the deck would still shuffle itself after being loaded in, causing ordered structure to be lost (.1 hour)

Day Total 2.6 hours

Sep 29th, 2018:

Fixed faulty logic which would cause builds not to load in their owner correctly (.5 hours)

Debugged issues with the build class having issues tracking its own value when its a multi build, and would be unabled to be captured depending on the order of execution

This issue was fixed by rewriting the system of how to store the numberical value, which is to have it dynamically calculated (constant time) when needed

2.5 hours of failed attempts to fix and debugging, .5 hours for current implmentation

Debugged an issue with decks having a segmentation fault when loading in an empty deck from the save file, (.5 hours to debug, .1 to fix)

Table now tracks the player who captured last, and gives all remaining cards to them (.1 hours)

Tournament uses pointers instead of a vector to track the round (.2 hours)

Added round end scoring to the table class, including multiple helper functions which calculate these values. (1 hour)

Restructed files to move files which were originally inline from the .h to the .cpp in multiple files (.5 hours)

Fixed output strings to be more verbose (.2 hours)

Allowed for build to be caputured by the other player and have the card properly unreserved from the other players hand (1 hour)

Added more output strings to meet the rubric requirements, and helper functions to get that data (.5 hours)

Day total: 7.7 hours

Sep 30th, 2018:

Added abilty to extend builds on the table, however the formatting in the multi build remains incorrect (1.5 hours)

Day Total 1.5 hours

Oct 1st, 2018

Updated the menu so the computer does not see the "Ask for help prompt" (.5 hours)

Added priority capture for the 10 of diamonds (.5 hours)

Fixed issue with white space preventing the next player from loading in correctly from a save file (.5 hours to Debug and fix)

Fixed bug where the human could make a build of zero table cards if they submitted blank input (.2 hours)

Created multiple new save files to test edge cases (.5 hours)

Commented all functions missing headers, and improved inline comments to be fully flused out (1.5 hours)

Added new function to builds to extract all the cards in them, this allows for builds to be properly be displayed (and counted) in piles for scoring (1 hour)

Day Total: 4.7 hours

Oct 2nd, 2018

Fixed an erronious >= (instead of >) which caused the AI to miss sets of 1 (.5 hours)

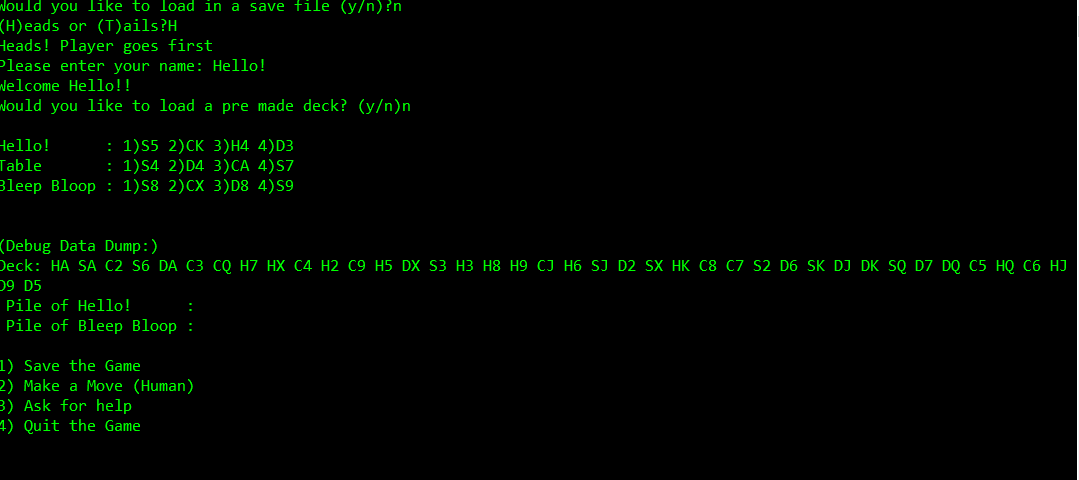
Improved generation of multi builds to prevent cases where data was lost, this code is similair to loading in save files(1 hour)

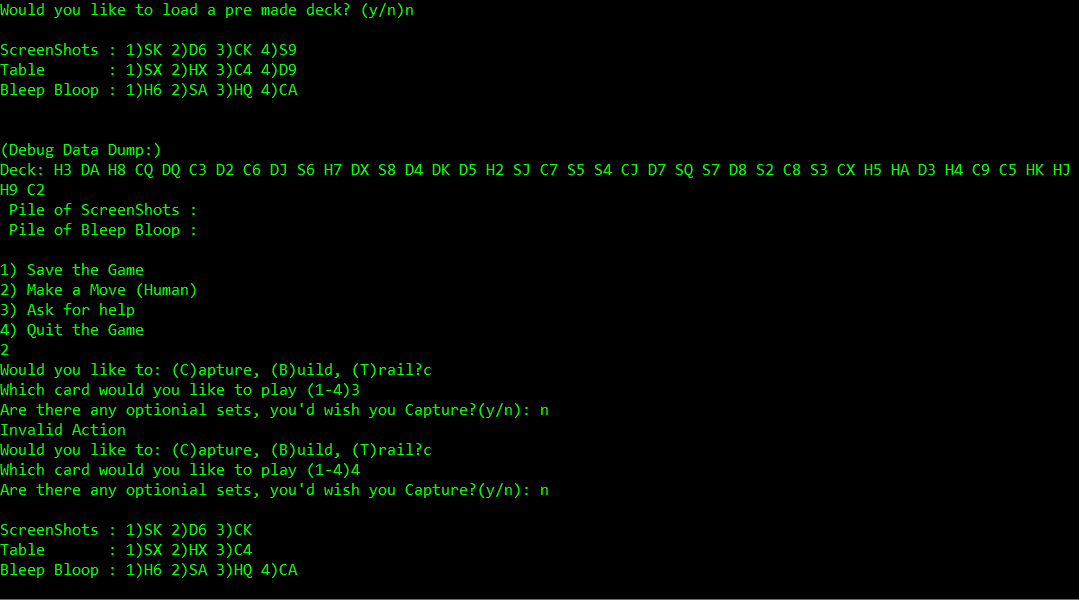
Improved UI, which includes a missed catch which had the Advisor speak as if it was playing, and improving output strings. Lastly, addedd a prompt for input before the program close (.5 hours)

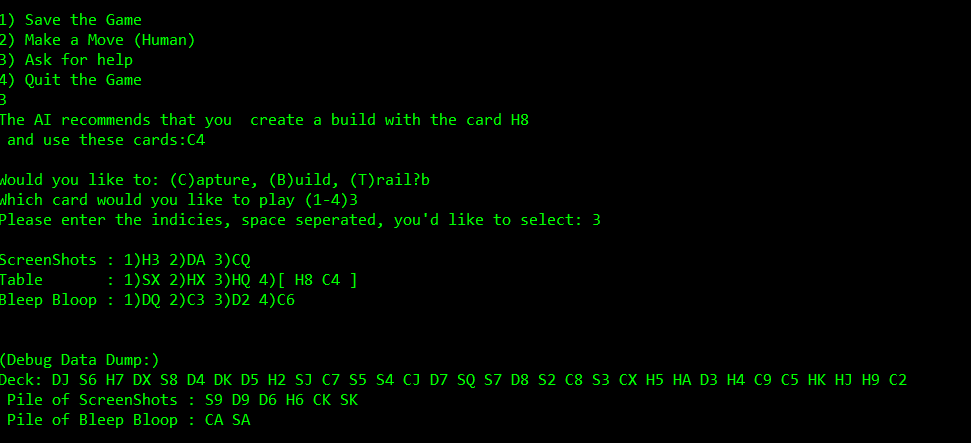
Day total: 2 hours

Project Total: 76.4 hours

# Screenshots







Capturing a multibuild

