Junior Honours Project

Module Code: CS3099
May 1, 2020

Group A-7:
Cristina Bikanga Ada (cba2)
Lei Ezz (hme3)
Ryan Hollreiser (rfh8)
Connor Denny O'Malley (cdo2)
Sam Stanford (sjs31)

Supervisor: Dr. Alexander Konovalov

School of Computer Science
University of St Andrews

Contents

Declaration

Abstract

The internet age has brought a surge of video games allowing users to play tabletop games across distances. This project successfully facilitates game-play of any trick-taking card game that can be expressed in a predefined language.

Our client offers a range of game-play options, including asynchronous and networked multiplayer games, as well as game-play with one or more Artificial Intelligence (AI) players. Our graphical user interface (GUI) allows for simple point-and-click interaction simulating the game-play of a real-life trick-taking game.

This report presents the aims and objectives of Group A7, how we designed and implemented our client, our adherence to software development methodologies, and the tools and techniques we used to promote communication between A7's members, as well as between A7 and other groups in our "super group".

1. Introduction

1.1. Background Information

Trick-taking games are ubiquitous and plentiful in the world of card games. A myriad of trick-taking games exist, from partners games like Whist (and it's many variations), to Hearts, where players go head-to-head individually.

Across these broad categories, the same trick-taking game can vary based on the country it's played in. Even the essential element of a trick-taking game, the trick itself, can vary in it's execution across games. This can be done, for example, with the introduction of bidding (such as in Bridge) or trump cards/suits (such as in Oh Hell or Spades).

There are many variables that allow us to differentiate across trick-taking games, offering such subtle changes that there are hundreds of combinations and variations that would result in unique games. Being able to easily play these myriad permutations as games would offer a great opportunity for people to engage in the culture of other countries, play games across great distances, or simply enjoy learning a new card game.

1.2. Project Aims and Objectives

In summary, we in A7 have set out to create a client program that allows its users to play any trick-taking game, so long as the rules of this game are given to the client in a language it understands. More specifically, we aimed for our final client to achieve the following:

- Collaborate with other groups within our "super group" to create a language for specifying trick-taking games, allowing our client to support game-play of any trick-taking game
- Networked play, which allows us to play specified games with other groups within our "super group"
- An intuitive, aesthetically pleasing user interface that allows our user to have a centralized locus of control and a memorable, stress-free experience
- The ability for the user interface to adapt to different games. For example, to identify and display cards in order of their strength, even if suit power/strength varies across games

- An Artificial Intelligence (AI) that allows the user to play against it in any game they would like with reasonable enough intelligence to entertain the user
- Asynchronous play, which allows users to play games over a long period of time

The given assessment criteria outlining our requirements was quite general to begin with, leaving us in supergroup A to collaborate and decide on our specifications together. The benefit of having such a vague spec was that we could convene and use the varied range of skills within us to allocate tasks we could flourish in creating. Our more specific design choices are outlined in the Design and Implementation section of the report, but the choices we had to make as a supergroup include deciding the language we would use and standards for representing games, the games we wanted to be able to play across groups, and protocols for compatible networked play.

The general specification also allowed A7 the opportunity to develop our thoughts/ideas in a way that was cohesive and grounded in existing information, as we had to do independent research to acquire sufficient knowledge about trick-taking games and the necessary technologies we would use to create our client.

As we advanced in this module, we kept the aim of this project close to heart: to prepare us for team programming at industry standard. Our goal was to team-engineer a piece of software under realistic industry conditions. It gave the students the freedom to set their own pace for inter-group communication and deadlines and allowed all teams to develop time management and to hone their skills in team programming.

Therefore, as we worked, we kept industry practices in mind and adhered closely to software development methodologies such as SCRUM and Agile(which were required for the module) and adhered to the principles behind these methodologies as strictly as we could. We also resolved to document our work closely and maintain regular inter- and intra-group communication to keep ourselves on track.

1.3. Proposed System

Because the spec we were given outlined functional requirements rather than technical requirements, our first decision as a group was to agree on what system to create. We wanted to think this through carefully given the aims and objectives of the project, as we wanted our end result to be versatile, portable, and reasonably conservative in its resource use.

We considered a range of options given our varied skill-set, as two members of the group were joint honors students, some had taken certain modules (for example, CS3105: Artificial Intelligence or CS3102: Data Communications and Networks) that would prove to be essential for programming certain aspects of the client.

Initially, we were inclined to create a web app that had a Java back-end. Our rationale was that a web app would be usable on any device so long as it had a web browser, meaning our program would be versatile and easily accessible. We thought this was important for a game client, especially a networked multiplayer one, as players are often on the go.

We were set on using Java from the beginning because it was the language we were all most familiar with, so we would save the time required to learn a new language. It is commonly used in industry, which is a benefit that lies close to the heart of the objectives of this module, as it is intended for us to gain experience working under industry conditions; and it is a very versatile language as it has been around for so long, so there are many libraries available to extend it's capabilities.

Another reason we decided to use Java is because we identified that Object-Oriented programming principles would aid us greatly in being able to create a product that matches the requirements and functions to our standards. For example, we agreed it was sensible to represent cards as objects, as it would give us a programmatic representation that made intuitive sense. In semester two, we agreed to use polymorphism to our advantage by using inheritance to represent our players, such that our game could treat all players the same, regardless of whether they were local, networked or AI players.

1.4. Success in Achieving Goals

We achieved all our goals with varying degrees of success. There is a more detailed evaluation in the evaluation section of this report. In brief, however, it can be said that our goals were modified retroactively and on a rolling basis. This is because of our adherence to the SCRUM methodology, which gave us the opportunity to reflect on our progress and to adjust our goals to keep as realistic as possible during our development.

A good example of this is our UI in our second SCRUM run (semester 2). We had intended to create an intricate and complex User Interface with interesting functionality, such as being able to adapt it's appearance and functionality to different trick-taking games. However, because of exceptional circumstances we had to revise our SCRUM plan in order to prioritize the base requirements of the spec. The change of plan required us to change our expectations for the project, which in turn led us to the difficult decision of changing the technologies we had wanted to program the UI in order to save time.

However, despite the atypical circumstances under which A7 was programming, we considered this project a success. Our rationale for this is as follows: this year-long project that we invested in was intended to hone our group programming skills, including communication, repository management, following SCRUM/Agile principles, and learning to create code that is easy to understand by other people.

Although we did not achieve all our personal technical goals, we are proud of this project. We managed to create readable, well-commented, modular and extensible code, all suited to Java's strengths as a programming language. Our code lives up to our standards for quality, and because it is modular, it was a pleasure for us to navigate the field of industry-style software engineering as a group.

We learned invaluable skills over the course of this year, such as managing expectations, communicating ideas accurately (both in person and using remote communication tools), and how to reasonably track our progress towards goals and prioritize tasks under time strain.

2. Design and Implementation

Our final hand-in currently supports 16 games. These are outlined in the appendix.

From the beginning, we had great aspirations for our project. We wanted to create a robust and modular project with loose coupling of components so as to maximize extensibility. This can be evidenced by the complete separation of the front end and back end, as the implementation for both of these is abstracted, yet functionality is accessible via interfacing.

2.1. A Language for Specifying Card Games

2.1.1. Deciding the Language

Before we could implement any gameplay, we needed to develop a language that would be able to distinguish card games intricately enough that our client could generate accurate game-play and rule adherence for a range of games. This language had to be consistent across the supergroup and employed as a standard. The aim for this language would be to define something specific enough that two groups given the same file in this language can smoothly play the game defined by the file together.

We as a supergroup initially considered using a context-free grammar for specifying games. This would be a flexible, yet sensible, option for what we were hoping to achieve. In particular, we endeavored to either create our own, or to use RECYCLE, which was defined specifically to describe card games algorithmically. RECYCLE boasts the benefit of accurate representation, as it's main feature is that it can represent the cyclical nature of card games through it's representation of player turns[1].

However, we had to be realistic and recognize our limitations. We recognized that time spent on this module would be limited by our other modules and that learning to create a context-free grammar or even to use a preexisting one (such as RECYCLE) would require an abundance of time. Since our time is a commodity, and we could not securely assume enough members in each group would be able to learn the technology, we thought it best to opt for something tried and true.

[1]RECYCLE

2.1.2. The Decided Language

This is why the supergroup chose to use JSON to represent card games. We chose to use the JSON data-interchange format as we have all had experience with it in the past, and because it is commonly used, a wealth of parsers and libraries exist to handle it's object parsing and creation. Using libraries such as Gson streamlined our development process, as it allowed us to serialize and de-serialize our representations of card games. This makes JSON extensible and straightforward to implement.

The trade-off here was that we lost the flexibility that a context-free grammar would give us, but with the time we saved from not having to learn an entirely new technology, the supergroup determined a JSON schema that could accurately represent trick-taking games. The JSON schemas less flexible nature also meant that some aspects of the specification had to be hard-coded, such as implementations for certain rules.

```
{
        "name" : "Two Player Whist",
        "description" : "A classic trick taking game, now
           playable if you have fewer friends!",
        "players" : 2,
        "deck" : null,
        "bid" : null,
        "teams" : [[0], [1]],
        "ascending_ordering": true,
        "can_view_previous_trick" : true,
        "initialHandSize" : 6,
        "minimumHandSize" : 0,
        "rules" : [
            {"name" : "leadingCardForEachTrick", "data" : "any
               "},
            {"name" : "nextLegalCardMode", "data" : "trick"},
            {"name" : "trumpPickingMode", "data" : "lastDealt"
               },
            {"name" : "calculateScore", "data" : "tricksWon"},
            {"name" : "trickThreshold", "data" : 6},
            {"name" : "trickWinner", "data" : "standard"},
            {"name" : "trickLeader", "data" : "prevWinner"},
            {"name" : "sessionEnd", "data" : "gamesPlayed"},
            {"name" : "sessionEndValue", "data" : 1}
        ]
}
```

Figure 2.1.: Our JSON representation of the game Two Player Whist. Another benefit of JSON is that it's key/value paired nature makes markup snippets very readable. If someone who had not known the format beforehand read this, they would likely still be able to figure out what each key/value pair meant for the game regardless.

2.1.3. Reading the Language

Game rules are passed over the network in the form of JSON schema. Instead of using a JSON Parser to read the schema, which would involve defining a rigid schema or building a complicated adapter, we have elected to read rules as input streams using GSON, going character by character until a full JSON object is recognized.

Reading the input character by character is more intuitive as it offers the flexibility of being able to read any JSON object, so long as its syntax is correct. If the schema for defining rules changes, we will not need to change our implementation, because it will still be recognized by our program. This leaves us open to changing game specs if we feel we need to without worrying about hard-coding solutions.

2.2. Product Design

Figure 2.2 shows the general structure of our final system, laid out graphically for ease of understanding. As shown in the figure, our project has been compartmentalized into sections. These are:

- **Back End:** Where the game functionality is stored, this is divided into three parts:
  - **GameLogic:** GameLogic is a package within our Java source that contains all the logic for games to run. This includes functionality for parsing games represented in JSON, bidding, turn-taking, game winning conditions/declaration, etc.
    It also contains object classes to represent real-world concepts pertaining to the game such as Cards, Decks, and representing players of all types.
    Finally, GameLogic also has functionality to support gameplay using a Command-Line Interface (CLI), if for whatever reason a user wishes to use it over the UI.
  - **Networking:** This package contains functionality for networking. This includes game discovery, game initiation/broadcast (to find other players), and the functionality to transmit and recieve moves across the network.
  - **Middleware (GO):** The pub-sub system is middleware that we implemented to allow for asynchronous playing, which we elected to be our special feature. This was written in Go and represents Games as topics, players as subscribers, and moves as messages. An HTTPS REST API connects the middleware to the Java part of the back end.
- **Front End:** Our UI was created in Python and is a graphical representation of a table-top. We build on HCI concepts such as showing system status to the user. It was implemented using the library PyGames, and interacts with the back-end through the use of Unix pipes.

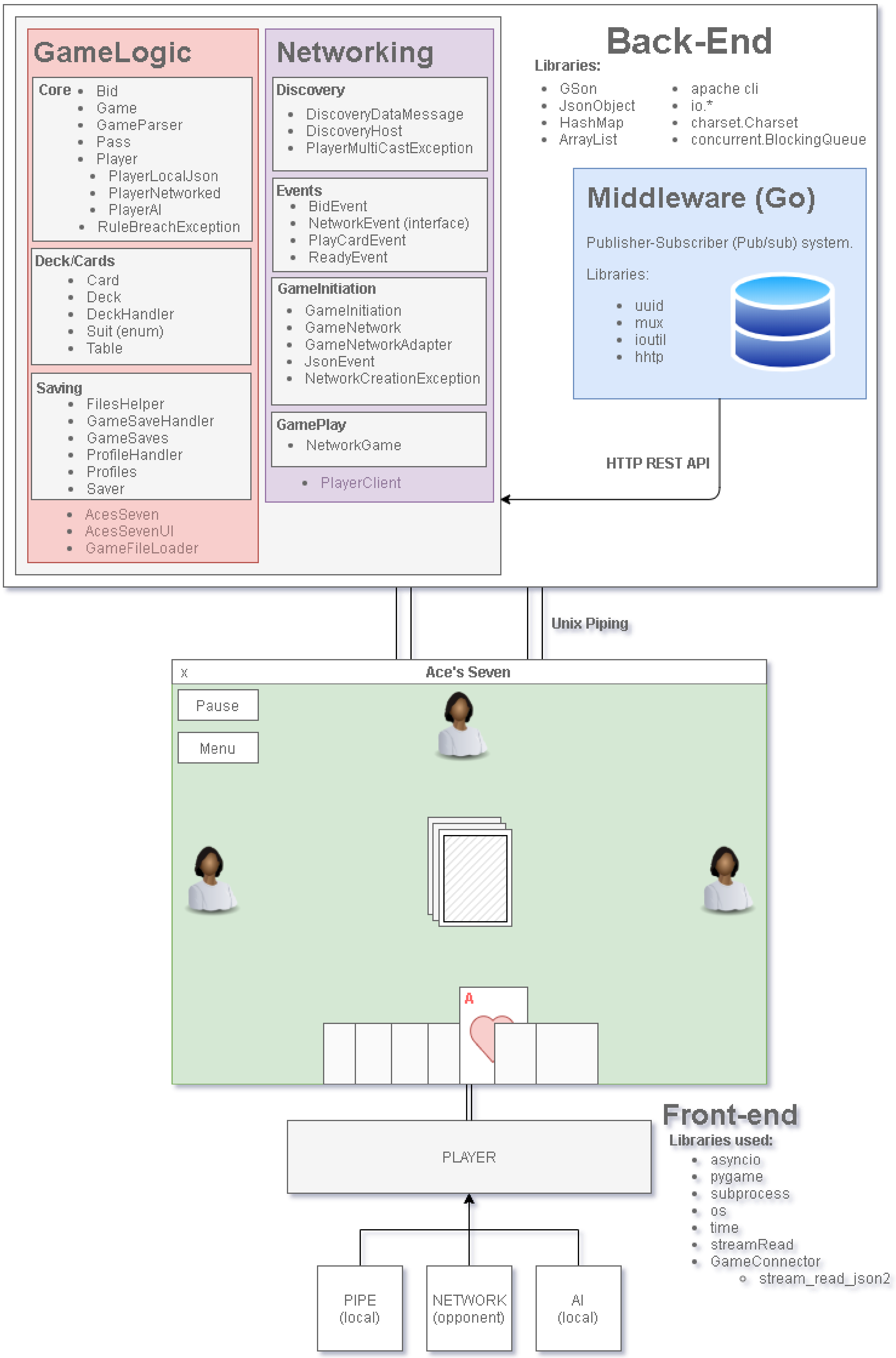


Figure 2.2.: A graphical depiction of our system's design.

2.3. Player Representation

Something noteworthy about the way that we programmed the implementation for the game is our use of abstraction. Within the **java.GameLogic.Core** package of the program, inheritance is used to abstract AI players, networked players and local (piped) players. The parent class Player is used directly, which means that all players are treated the same regardless of what kind of player they are.

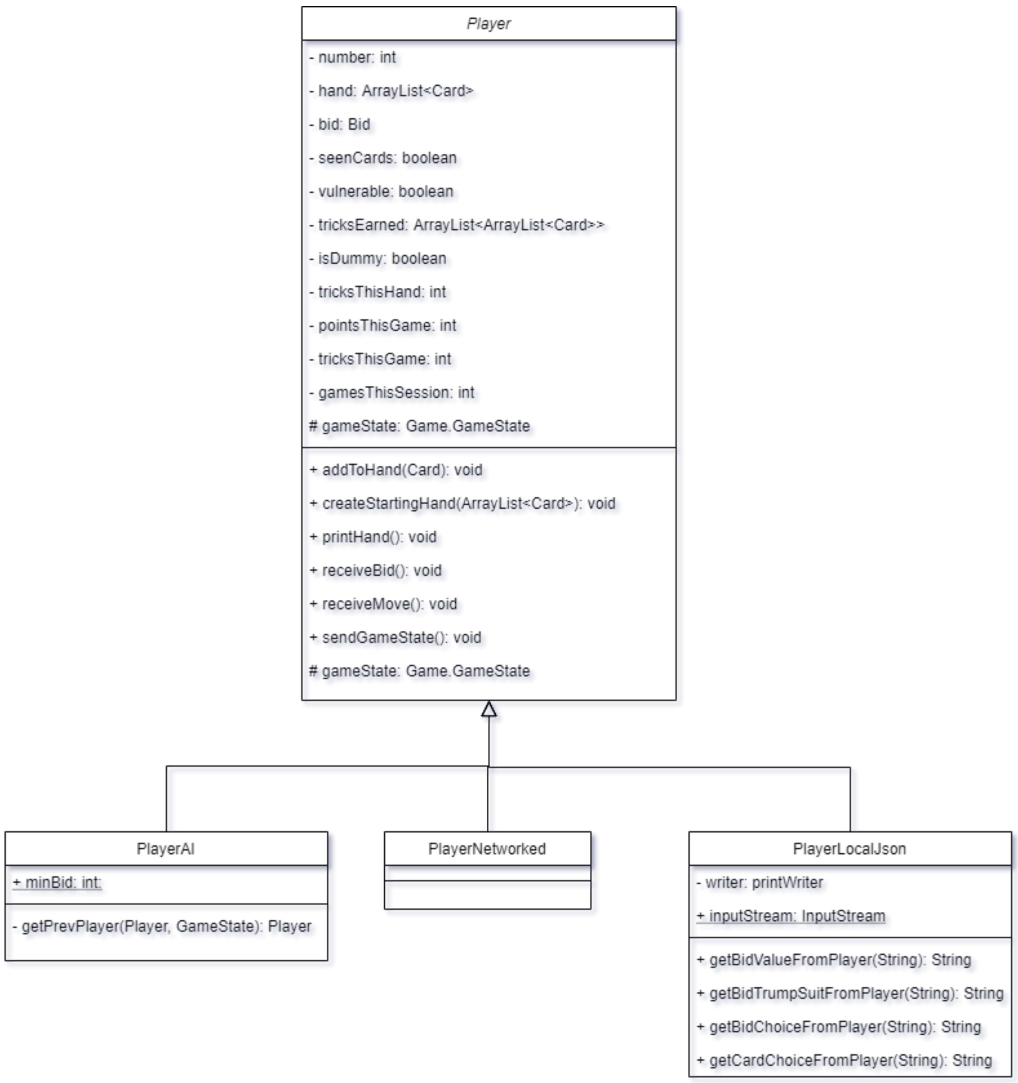


Figure 2.3.: A simplified UML diagram showing the inherited implementation of the Player class.

The reason this is a good thing is it indicates loose coupling of code components. This means we can use dependency injection to substitute different players in without having to adapt the game to support them.

If we wanted to add or remove types of players, nothing in the game logic would need to be changed so long as the players were child classes of the Player parent class. This also makes our project modular and easily extensible, and it makes it easier to test as we can organize games between multiple AIs, multiple local players, or any other configuration of players we would require to rest. Multiple AIs in particular is helpful for testing as we can put AIs to play against each other to test our implementation without spending too much time.

The separation of concerns that dependency injection results in allows for greater code readability, reuse and modularity, which is paramount in the context of group programming. This meant different members of the group could work on different Player types at the same time with the security of knowing their implementation would be interchangeable in the eyes of the program.

2.4. Ensuring Accurate Gameplay

We have included a number of custom exceptions in the Game Logic part of our program in order to handle specific things that could go wrong in the game. These exceptions extend the standard Java class Exception.

Our **RuleBreachException** is used to signify when a player breaks the rules. If a rule breach is detected, the game state will not accept the response provided by a player. It will throw this exception and then either ask for another move if the game is local, or intentionally cut the connection if the game is over a network.

2.5. Deck/Card Representation

In proper Object-Oriented fashion, we have chosen to represent certain entities in the scenario we are modelling (card games) as objects. In this section, we will outline the main object classes that are described in the **GameLogic.DeckAndCards** package, as well as interesting characteristics about their implementation.

2.5.1. Card

This class represents a single card. It uses an enum, Suit, to ensure that the suit type of each card is always valid and fits one of the four french suits (Spades, Diamonds, Hearts and Clubs).

The implementation of this class is noteworthy for a number of reasons. Firstly, we have chosen to represent card values as integers. This makes it easy to compare the strengths of cards, a functionality which is the underpinning essence of trick-taking games. To compare cards, we have employed polymorphism to override the object class's inherent "equals" method, which compares cards by performing a sort of 'deep copy' instead of comparing object references, which is what the default equals() method does.

Another interesting thing about this class is the implementation of conversion methods that allow card values to be converted to/from Strings and ints. This is because the supergroup implemented their card representation as Strings, which meant we had to adapt our code to be compatible with theirs. However, this was a blessing as it allowed for greater flexibility in our code and meant we could apply custom configurations that are difficult to implement using strict int rules.

Our program abstracts the String name of the card (used by the supergroup) from the JSON rules into an integer, which allows for easy comparison and reduces errors when programming.

2.5.2. Deck

Our Deck class is essentially a custom implementation of a stack, using Card objects. We used an ArrayList of Cards to represent the deck, and programmed our own stack methods for it:

- **push(Card card):** Pushes Card onto the top of the deck.
- **Card card pop():** Pops the top card off the deck, removing it from the deck.
- **Card card peek():** Allows the user to see the top card on the deck, but keeps the card in the deck.
- **burn():** Removes the top card from the deck, but does not return it.

2.5.3. DeckHandler

This class does not necessarily represent a concrete, tangible concept, but is just as important as the other classes in this package. This class contains the implementation for the shuffle, which uses the Fischer-Yates shuffle:

```
for i from n-1 down to 1 do
  j ← random integer such that $0 \leq j \leq i$
  exchange a[j] and a[i]
end for
```

The method for this shuffle, which we agreed to use as a supergroup, can take a variable number of cards in a variable amount of decks and shuffle them. The importance of this is that it makes the method extensible. Therefore, it is easy to apply to games with special decks. A consideration we have taken in this class regarding the shuffle is that it is not efficient when there is just one deck, as it builds two packs of cards. That means the memory usage is redundant for only one deck.

2.5.4. Table

This class represents a table, more specifically the cards that are piled in the middle of a table during a game.

2.6. Bridge

Perhaps the most difficult part of our implementation was programming the game Bridge to work. This was a necessary functionality for our program, as the spec required us to be able to play this game across our supergroup.

There are a number of reasons that Bridge was a challenging game to implement:

- One "game" of bridge consists of multiple components: The auction (bidding), play (trick-taking), and scoring.
- Bridge is played as "best out of three", meaning that the above components have to be repeated three times.

There were also a number of things we had to change in our representation of the game due to technical concerns, such as the "dummy hand". Typically, in Bridge, after the bidding round, the player who wins the auction (known as the Contract Declarer) plays on behalf of their partner, known as the "Dummy".

However, as we had already implemented networking functionality for our project by the time this requirement was given to us, we elected to ignore this part of the rules because being able to program our network protocol to accept players behaviors being "not in turn" or for one player(the contract declarer) to play twice (once for themselves and once on behalf of the dummy) would have required a complete overhaul of our code and a level of complexity that we agreed as a supergroup we did not have time for due to the extenuating circumstances.

Another problem we faced when programming Bridge was with our JSON specification. Because of the way we had agreed to represent games as a supergroup, representing Bridge using JSON involved a lot of code inconsistencies. For example, there is an array of key/-value pairs called "rules" in the bridge spec, which contains rules for the game. However, rules are defined elsewhere in the JSON implementation, namely in the bidding block. This was something we considered an oversight on the supergroup's part, as we discovered our JSON spec was not versatile enough to represent the game of Bridge without being inconsistent in it's structure.

The implementation of this game is overall poor in our opinions, and we are not incredibly satisfied with it. The supergroup representation of the game was not as neat as we would have liked, and because of this alongside our other extenuating circumstances we feel we did not meet this requirement to the best of our ability.

2.7. Networking

2.7.1. Functionality

While the most intuitive solution to offering networking functionality would have been to form networks using a star topology, offering a centralized, non-biased server that could enforce rules and prevent cheating by disallowing any one group to hold power over it's execution. However, because we are not allowed to share code across groups, we recognized that we had to do it another way.
When implementing the networking functionality, we kept two main things in mind. These were equality of nodes, meaning no centralized power could exist during networked games, and the assurance that we could communicate with other groups reliably. Both of these issues were solved with the discovery protocol, which provides versatility and allows trick-taking games to be broadcast over a network.

2.7.2. Discovery Protocol

Created in first semester by one of our members, the discovery protocol outlines the process for communication between clients to set up a networked game. The implementation for this protocol can be found within the **Network.Discovery** package in our Java source folder.

We considered basing this protocol off the paper "Secure Trick-Taking Game Protocols" (Bultel and Lafourcade, 2018) to prevent cheating in the game, because we thought that it would be easy for other groups to program their clients to implement cheating if we used a server-client topology in the program. This is why the discovery protocol initiates a mesh network topology between nodes.

The key feature of this is that a mesh network is created as a result of this process, meaning there is no centralized "server" as such after a game is initialized, but rather that all nodes in the network are connected to each other in a peer-to-peer fashion.

It is also important to note that one of our main goals in designing the Discovery protocol was to prevent cheating, but our specification was later on updated to specify that we did not need to do this. This is because it would be nearly impossible to develop something that prevents all forms of cheating from other groups when so many different technologies and languages are being used to program the clients in our group.

So going forward, we agreed not to check that other player's games were properly working, but rather to assume that incoming moves are correct. This better reflects the workings of a real-life card game, as players can cheat and go undetected.

However, to also reflect a real life card game, game clients need to keep track of the distribution of cards to ensure players are not using cards that they could not possibly have (ie. "injecting" duplicate cards into the game). This means it's up to the game logic implementation, rather than the networking implementation, to identify cheating.

2.7.3. Game Initiation

To host a game, our DiscoveryHost class sends out a datagram beacon on UDP multicast following our protocol, which required datagrams to be sent in a specific format:

```
WHIST:n:m:10.143.121.203:4444:Tom
```

N is the current number of players that have connected to the host, so this represents how many players are required to fill the game. M is the total number of players in the game.

The host counts as a player for the purposes of counting this. The last field is optional, and corresponds to the host's username.

UDP multicast is used for the discovery protocol as members of the supergroup that took networking were most familiar with this method of multicast. The benefit of UDP multicast is that it sends a request to the router of the network instead of to each client machine. This is less congestive for the network being used, which we found to be of importance as all groups were intended to test on the Computer Science network, and we wanted to run our programs quickly and efficiently over the network.

2.7.4. Networking Exceptions

We created our own custom exceptions to aid in our debugging, as we understand it is imperative for errors to be explained as specifically as possible. This only applies more with networking errors, as there are a whole host of things that can go wrong.

The exceptions we have made are:

- **NetworkCreationException:** This alerts the program if there was an issue in creating a network. Reasons this exception could be thrown are things like players dropping out during the discovery beacon or a player's connection no longer being active while the program tries to connect to them.
- **PlayerMulticastException:** This exception is thrown if the program cannot connect to a multicast group(join a game). This would be more likely to be an issue with the player's router or internet connection rather than with the game clients themselves or the discovery protocol.

2.7.5. Events

In the **Network.Events** package, we have a set of classes representing events. Events are used during gameplay and passed back and forth over the network to communicate data from players.

The event classes we currently have in the project are as follows:

- **BidEvent:** Communicates bids (value, suit, etc)
- **PlayCardEvent:** Communicates cards (type, suit, rank).
- **ReadyEvent:** Communicates that a player is ready to play.

2.8. Graphical User Interface (GUI)

By semester two, three of the five members of A7 had been trained in Human-Computer Interaction (HCI) concepts due to the eponymous first semester module run by the university (CS3106). Once we had taken this course, we took it upon ourselves to design a smooth, intuitive GUI using the principles we had learned as a basis for it's flow and structure. Considerations included Neilsen's ten heuristics and the foundations of mobile-first design.

One of the requirements of this assignment was to have a functional and exceptional User Interface. This section of the report will walk through the technologies and strategies that we researched, explain what we wanted to achieve had we still been geographically proximal to each other, and detail what we actually achieved in terms of the UI. We will also explain how this was done.

2.8.1. Heuristic Evaluation

Before designing the product, we considered what specifically we wanted users to draw from their experience playing the game. We wanted a game that was intuitive, seamless to play, and enjoyable to interface with. In order to identify how to do that, we consulted Nielsen's 10 heuristics of user interface (UI) design and kept them in mind during the design process.

After our research, we evaluated the following to be our nonfunctional requirements for the system:

- Do not force users to use their memory. System status should be displayed on the screen at all times so the user does not have to keep anything in mind while playing games.
- Keeping the design minimal. We endeavored to have a simple, non-confusing design that utilized the functionality of menus and drop-downs to the fullest, so that the user would only have to focus on the most important things at any one time.

- Matching the product to the real world and maintaining consistency of jargon. This means terms that are familiar to the user should be used – for example, "Start Game" would be used instead of "Initiate Discovery Protocol to Create a Peer-to-Peer Network for Playing A Game", as the user does not need to be aware of the implementation of the functionality as that would just add confusion.

- The user must have a centralized locus of control. This is because we want the system to feel natural to use. It was important to us that the user felt in control of the system flow at all times, and we endeavored to make a UI so seamless that the user felt like they were controlling a game rather than interacting with a computer screen. We wanted the UI to bridge the gap between the game and the user, rather than the program and the user.

2.8.2. Technologies Considered

In the beginning of this stage of the product, a subset of our team met regularly to discuss options for the UI. We intended to exercise our web development skills and create a front-end to allow for versatility and portability of the application.

However, midway through the second semester we had to readjust our plans as outlined in our COVID-19 Plan. The UI undertook what we would describe as a "role switch", that is, another member of our team took over heading it, as we did not have enough time to learn the technologies that would have been required to implement the client as a full stack app, but this other member was proficient in the Python library PyGames.
This person took over the development of the UI, using PyGames to create a front-end that was connected with the back end using Unix pipes.

Mobile-First Design

To continue the trend of employing industry standards in our programming and group work, we endeavored to use modern techniques in our UI design. The main web design/development technique we used is the mobile-first approach, which suggests that designers should consider mobile screen sizes first, and then scale up to larger desktop screens by using responsive web design.

The main advantage of this is that important content is not shrunk or removed for mobile users, but is instead displayed in creative ways - with over 50% of internet users being on mobile[2], this is crucial for our application.

As shown by figure 2.4, we considered the way that a trick-taking game would look on a mobile screen before all else. Keeping the heuristics in mind as we did this, we aimed for a minimal, simple design, using concepts the user was familiar with (such as drop-down

[2]Statista, 2020

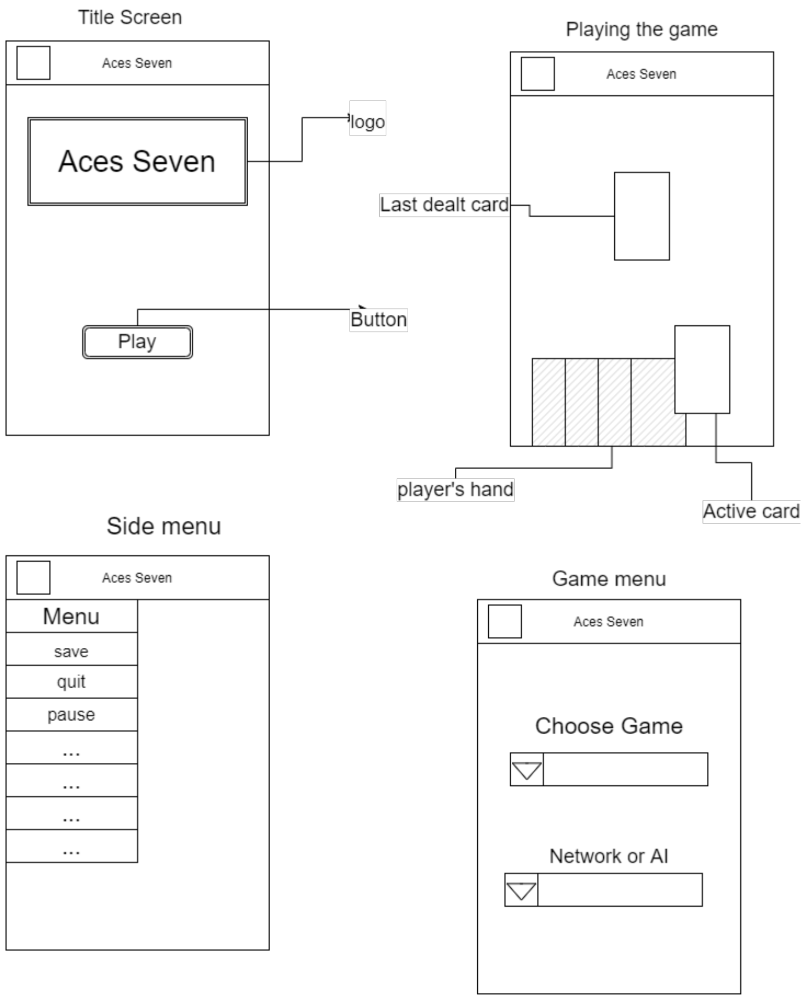


Figure 2.4.: A drafted image of the general layout of the program, following Mobile-First design principles.

menus, graphical depictions of cards, and the classic "sandwich" menu button often found on mobile apps).

Lean UX

Keeping in accordance with our goal to use industry-standard design techniques, we wanted to use Lean UX. This strategy focuses on shifting UX design and development to follow Agile software development principles.

We based our process around the principles discussed in Lean UX: Applying Lean Principles to Improve User Experience by Jeff Gothelf and Josh Seiden, which included a rapid, iterative design process which involved as many of the team members as possible, as well as using stand up meetings and retrospectives.

2.8.3. Technologies Used

Although we had strong intentions of using specific technologies and following Agile procedures to create our UI, the setbacks we experienced due to COVID-19 forced us to change our plans quite quickly.

We organized ourselves within the span of a week and restructured our entire plan for the GUI. Now our objectives were more to make it functional rather than to make it exceptional, as we understand at the end of the day that it is ore important to have a functional product than separate exceptional pieces.

Python and PyGame

We elected to use Python to design our UI. This was for a number of reasons: most importantly that we all had Python experience anyway. Python also is a well-documented and versatile language with a wealth of resources and libraries available at programmer's disposal.

The most noteworthy of those which we used is PyGame, a library designed to be used by game developers. The benefit of this library is that it comes with a number of methods that aided us in our goals: namely, it allowed us to represent cards graphically by representing them as sprites, and to group cards graphically by representing them as sprite groups (for example, a player's hand would be expressed as a sprite group).

PyGame is also highly portable, which aligns perfectly with our goals to create an extensible and loosely coupled product. This meant our UI could run on many distributions of OS, including Linux, OSX and Windows, and can run using graphic backends such as DirectX and OpenDL.

Last but not least, as a gaming library, PyGame comes heavily documented with a plethora of visual tutorials and guides which enabled us to troubleshoot very smoothly.

Unix Piping

We used Unix piping to connect the front end to the back end of the project. The reason we did this is to follow Unix Philosphy, the second point of which states that the output of one program should be the input of another. Pipes embody this ideology.

Within our program, the use of Unix piping is bidirectional, meaning that the front-end and back-end draw from each other for their input and output.
The main benefit of this is that it is language-independent, meaning that we could build the UI in whatever way we wanted to, so long as the interfacing was correctly configured.

2.8.4. Final User Interface

We elected to use the Pygames Python library because it is accessible, extendable and easy to use. Moreover, Python was easy to integrate with the logic and JSON that the back-end of the project works with. While React is also accessible and integrates well with the back-end, Pygames was build as a system for coding video games and thus had a handful of tools that aided this process that React didn't have built in.

We have been developing the framework for the systems in the game, adding start menus, pause menus, and end screens. In addition, we have written code that draws, renders and animates assets, as well as code that creates messages and a HUD for the player.

2.9. Artificial Intelligence

Our Artificial Intelligence works for a number of games, but unfortunately not for Bridge. We did not have high expectations for this part of the project because to write a program that can validly play **any** trick-taking game is quite time-consuming and outwith the scope of this project.

We made an AI that randomly chooses valid moves to play Two-Trick pony and Whist, keeping track of it's bidding and move-making. The production of our AI was made this term following a major overhaul of our project including massive refactoring in order to allow the AI class to extend the player class, as mentioned before, making good use of abstraction.

Although the AI is not intelligent in that it does not learn from other player's moves, the process we underwent to create the AI involved collaborating as a group, implementing the Continuous Integration in our repo, and a few other changes that brought our product up to Industry production standards. Because of this, we see our shortcomings in the AI as a learning experience that better taught us to group program and manage expectations.

2.10. Special Feature: Asynchronous Playing

We were required by the specifications to implement a special feature in our client that could not be done in a real-life trick-taking game. Inspired by asynchronous games such as Bike Race and DrawSomething, we endeavored to create a publisher/subscriber system(pub/sub) that would allow players to play games asynchronously.

2.10.1. Why GoLang?

We decided to use the language Go to implement this. This is because Go is used by many big companies to implement middleware, and has a solid reputation as it is a language created by Google developers. With this in mind, we used it to make a robust and straightforward middleware, which we used as the backbone for our asynchronous play function.

2.11. Game Saving

We implemented the game saving functionality using Java. Two files in particular were used for this: GameSave and GameSaveHandler. GameSave was more of a helper class, whilst GameSaveHandler provided the bulk of the functionality:

- **saveGame()** is the method implemented to handle converting game state attributes (for example, players, player hands, etc.) into an object that is written to a JSON file using the Jackson library.
- **loadGame()** does the opposite, using the Jackson library to read JSON format and parse it into a game state. The drawback of the way that we programmed this is that due to some limitations with the Jackson library, some attributes in the JSON files have to be read in as Strings and then converted into variables that can be used by the Game Logic. This method returns a Game State, which can then be used by the middleware pub/sub to load up a game.

3. Software Development Methodology

3.1. Initial Plan/Structure

3.1.1. Functional Requirements

The beginning of our project time was spent planning and detailing what would be done throughout the oncoming weeks to maximize the amount of time spent programming throughout the semester.

3.2. SCRUM/Agile Methodology

3.2.1. Why SCRUM?

Scrum is not always the best methodology to use when developing a program. It is an agile method and thus, if a system has a very detailed specification or is subject to some kind of external regulation a plan-driven development approach may be more appropriate. However, beyond the fact that it has been mandated for this project, the Scrum approach is very well suited to the development of this program. Scrum is best suited to smaller scale projects with close-knit and geographically close teams that will have no difficulties with communication working on a program with a short expected lifetime.

Using the Scrum approach enabled the team to make iterative improvements to the program via short-term, graspable goals in the service of our long-term aims while also creating natural stopping points in which the main stakeholder (the team's supervisor) could be consulted. Moreover, using Scrum methods affords each member of the team the ability to take ownership of the code and ensures an equal distribution of work across all of the members. Given all of these advantages, it was clear to us that the Scrum approach to development is well observed and must be followed by the team in the creation of this program. To that end, the team agreed to meet twice weekly for sit down meetings to ensure that any issues in development can be highlighted and addressed immediately.

In addition to this, the team agreed to have digital standup meetings every day, with the contents of these meetings recorded in a group repository on Google Drive. These frequent meetings make sure that all members have visibility of the whole project and to guarantee that all tasks in the sprint cycle are accounted for and prioritized appropriately. The team has agreed on two-week long sprint cycles. This duration is on the shorter end of what is typical for Scrum development, but is still fairly standard.

The team decided on two-weeks because it was determined that this was enough time to accomplish a meaningful amount of tasks. This duration also allowed for more frequent review and assessment periods which the team decided would serve to keep the project organized and on-track.

3.2.2. Our use of SCRUM

We were obliged to consider the deadlines we had for our other classes and plan around it. For example, during Week 6 we were given quite a number of deadlines and we took that into consideration when planning the sprint that coincided at that time.
The waterfall nature of the SCRUM methodology has proven to be a good fit for our team. The iterative approach has the benefit of being vague enough that we can work at a variable pace and still be on track, since our objectives are generalized and not incredibly specific, our day to day work is not as important as our weekly work. Each member in the team can work at a pace that will allow them to complete their goals, so long as it happens within the time-frame of each sprint cycle.
A number of setbacks affected the velocity of our burndown. Most notably, of course, the pandemic. Additionally, when designing our original SCRUM plan, we'd agreed on having stand-up meetings every day. This made sense at the time because our SCRUM sprints were 2 weeks each, and we perceived it realistic to work on the project every day.
We realized quite quickly that the wide difference across our schedules would mean our stand-ups would be less often than we'd planned. We are five students taking a range of classes, some of us being double majors. All five of us could not reasonably spend time working on this project consistently (daily), and at the same time as each other, because there was always one of us indisposed due to a deadline at some point throughout each sprint. Therefore by semester two we decided to use more general sprint plans and implement weekly standup meetings, which prove to be much more sensible.

3.3. Intra-group Communication and Task Management

We aimed to meet weekly as a group, but because of deadlines, different majors, extracurriculars and even supergroup meetings, it was not possible for us to properly arrange a regular meeting time. We were able to meet often as a group, but when we could not meet as a group of five, then people who had questions would meet in-person in a smaller group to sort things out.

This often took the form of people who were doing the UI meeting with people who had worked on the game functionality, or in the case of first semester it was often the members who arranged the networking functionalities with those who were implementing the actual gameplay.

Because we did not get the luxury of regularly meeting in-person, as one would in the software engineering industry during actual SCRUM, we had to supplement our interactions using a number of tools and technologies to convey the work we'd done and any questions/queries we'd have, as well as allocating tasks.

3.3.1. In-Person Meetings

For the first 6 months of the project, we had regular in-person meetings both within our group and with the supergroup. Verbal communication is crucial in team programming as it gives us the ease of being able to explain concepts rapidly and clearly. Meeting in person also allowed us to use the resources available to us in the School of Computer Science, including the tutorial rooms complete with screens big enough to present with as well as plenty of whiteboards to express abstract concepts, brainstorm and write out plans.

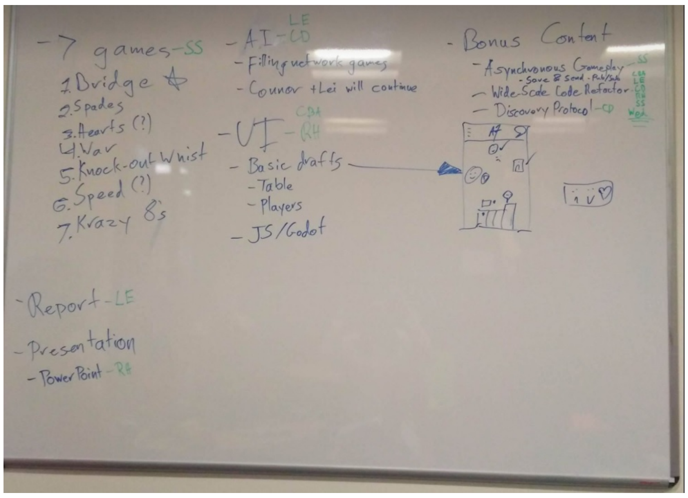
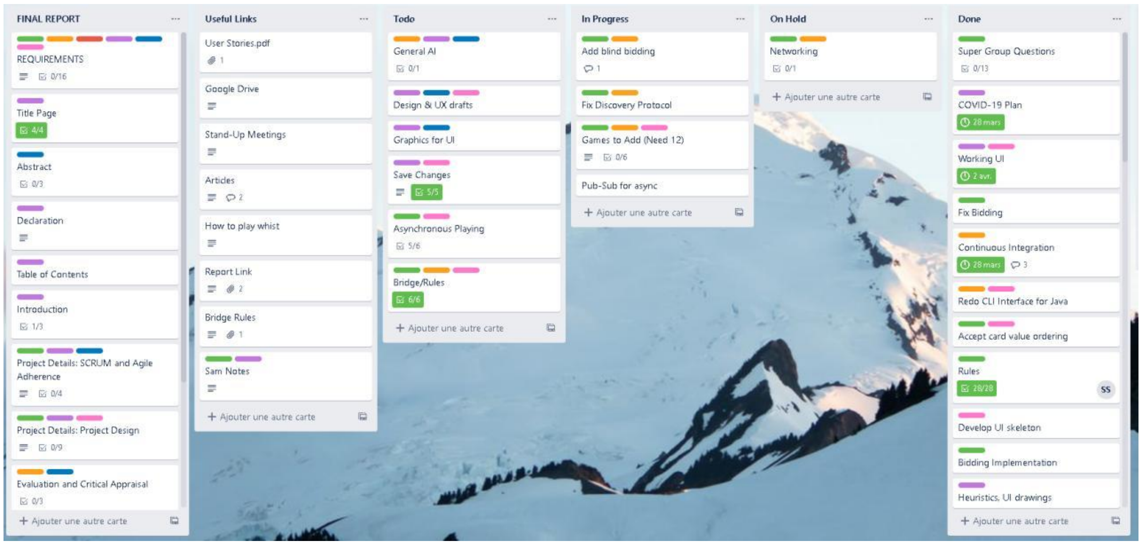


Figure 3.1.: An image of a whiteboard we used during a meeting on the 2nd of March.

3.3.2. Trello


FINAL REPORT

- REQUIREMENTS 0/16
- Title Page 4/4
- Abstract 0/3
- Declaration
- Table of Contents
- Introduction 1/3
- Project Details: SCRUM and Agile Adherence 0/4
- Project Details: Project Design 0/9
- Evaluation and Critical Appraisal 0/3
- + Ajouter une autre carte

Useful Links

- User Stories.pdf 1
- Google Drive
- Stand-Up Meetings
- Articles 2
- How to play whist
- Report Link 2
- Bridge Rules 1
- Sam Notes
- + Ajouter une autre carte

Todo

- General AI 0/1
- Design & UX drafts
- Graphics for UI
- Save Changes 5/5
- Asynchronous Playing 5/6
- Bridge/Rules 6/6
- + Ajouter une autre carte

In Progress

- Add blind bidding 1
- Fix Discovery Protocol
- Games to Add (Need 12) 0/6
- Pub-Sub for async
- + Ajouter une autre carte

On Hold

- Networking 0/1
- + Ajouter une autre carte

Done

- Super Group Questions 0/13
- COVID-19 Plan 28 mars
- Working UI 2 avr.
- Fix Bidding
- Continuous Integration 28 mars 3
- Redo CLI Interface for Java
- Accept card value ordering
- Rules 28/28 SS
- Develop UI skeleton
- Bidding Implementation
- Heuristics, UI drawings
- + Ajouter une autre carte

Figure 3.2.: An image of our Trello taskboard. Labels are color-coded to indicate task responsibility.

We elected to use the digital taskboard Trello to keep track of our tasks. We were recommended by the module coordinator and our lecturers to use a physical taskboard. However, there were a few considerations we had to take into account:

- **Where to store the taskboard:** We did not want to keep it in a public space such as the labs, because then it could potentially be lost, tampered with, or damaged when left unattended; we also did not want to leave a physical task board in any of the member's homes, as that would make it difficult for the other members to access. An online taskboard is easy to access from any location/device, and the added benefit of Trello specifically is that it has an app.
- **Accessibility:** The taskboard needed to be easy to reach and easy to modify and add to. Having a physical taskboard would mean that alongside our devices, we would need to have pens, sticky notes, etc. to complete this project. A digital taskboard allows us to add to, modify and delete cards on the taskboard without using exhaustible resources such as ink or paper, and is easier to add to than a physical taskboard as most of us can type faster than we can write.
- **Extra Features:** We found it more sensible to use Trello because the online taskboard came with a multitude of extra features that would prove to be helpful in our SCRUM adherence, including:
  - - Colour-coding tasks
  - - Creating deadlines with e-mail notifications
  - - Dynamic Checklists
  - - Attaching files to tasks

3.3.3. Version Control Using Gitlab

Each group was given access to a GitLab repository to host our code and collaborate. Gitlab is a version control technology that uses git as it's basis, and it and programs like it are used ubiquitously in industry.

The benefits allowed by GitLab are as follows:

- **Managing dependencies:** We were able to install Maven directly onto our repository to be able to manage dependencies.
- **Continuous Integration:** We created tests that could run on pipeline, which would allow our master branch to be secure in it's functionality at all times.
- **Remote Work:** Using ssh public keys, we set up our repo to be accessible from our home machines. This turned out to be useful when we became geographically distributed due to the virus.

- **Change Tracking:** We were able to handle merge conflicts easily with the security that we could simply roll back to a previous version if our merged branch failed to meet the pipeline tests.
- **Comments on Merges:** Being able to provide documentation or a short description of the merges we made as we made them allowed our group to track our progress of the program's design and build as well as to allow us to better understand each other's implementation and accomplishments.

3.3.4. Continuous Integration/Maven

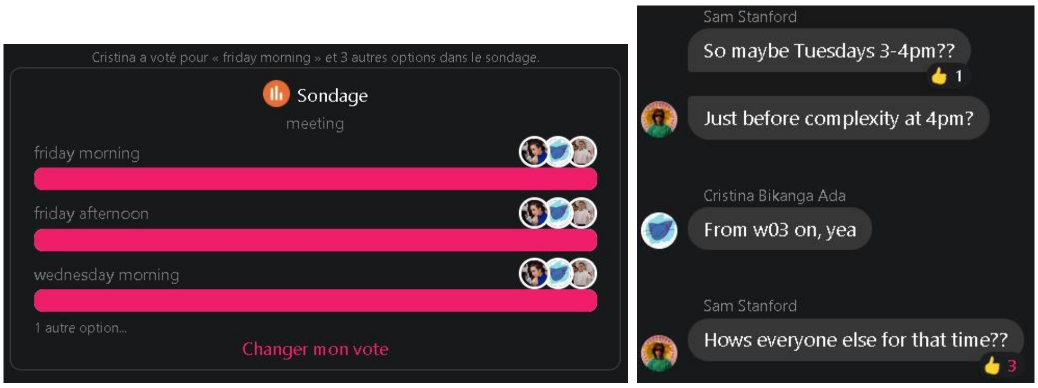
It was important to us that our master branch on gitlab only contained fully functional code at all times, and we wanted to maintain a sense of consistency in our code. We thought the best way to do this was to introduce pipelining, which will not allow our merge requests to pass unless the code we are trying to merge passes a series of predefined tests. If it does not, then it can notify us of this in a number of ways, making it a flexible tool for remote work.

The code was organized into packages to make it easier to navigate. Previously we would manually download dependencies and add them to the class path. To streamline this process, we have setup Apache Maven. Maven automatically downloads the dependencies and includes phases for building, and automatically running tests. Tests are found by maven if they adhere to a naming convention and are run before building. Maven works well with the pipeline because it packages our program easily and runs our tests automatically, even as new ones are added. If Maven cannot build the program, then the pipeline fails, blocking a merge request.

Maven also automatically produces an executable .jar in the build phase to make for easy distribution. By being able to automatically run tests Maven can be used with the Gitlab continuous integration system to require tests to pass before merging branches.

3.3.5. Facebook Messenger

We stuck to using Facebook Messenger for instant interaction throughout the year. Messenger is an app that we all have on our phones and use regularly anyway, so team members seeing a message quickly after it's been sent is likely. We can also send attachments where needed, but since Messenger formats hyperlinks, we often linked directly to cloud documents when we needed to reference them. Another benefit of using Messenger is that we can organize meetings at a moment's notice.
However a disadvantage is that Messenger is attached to the social media site Facebook and thus its notifications on non-academic topics can distract when doing work.
After COVID-19 emerged, we used Messenger more frequently and specifically its video and audio call functions, as a way to do group meetings before we got used to Microsoft Teams.



Cristina a voté pour « friday morning » et 3 autres options dans le sondage.

Sondage
meeting

friday morning
friday afternoon
wednesday morning
1 autre option...
Changer mon vote

Sam Stanford
So maybe Tuesdays 3-4pm?? 👍 1

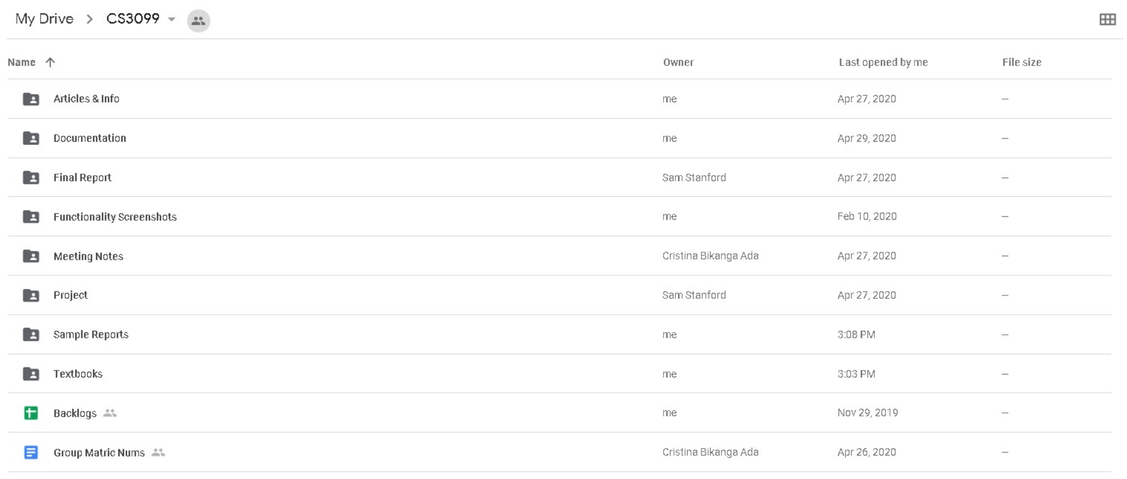
Just before complexity at 4pm?

Cristina Bikanga Ada
From w03 on, yea

Sam Stanford
Hows everyone else for that time?? 👍 3

Figure 3.3.: The convenience and simplicity of Messenger's features allows us to set up meetings quickly and with low hassle.

3.3.6. Google Drive



My Drive > CS3099

Name	Owner	Last opened by me	File size
Articles & Info	me	Apr 27, 2020	–
Documentation	me	Apr 29, 2020	–
Final Report	Sam Stanford	Apr 27, 2020	–
Functionality Screenshots	me	Feb 10, 2020	–
Meeting Notes	Cristina Bikanga Ada	Apr 27, 2020	–
Project	Sam Stanford	Apr 27, 2020	–
Sample Reports	me	3:08 PM	–
Textbooks	me	3:03 PM	–
Backlogs	me	Nov 29, 2019	–
Group Matric Nums	Cristina Bikanga Ada	Apr 26, 2020	–

Figure 3.4.: Our shared file repository on Google Drive.

We have implemented the use of the cloud storage system Google Drive to store and organize our project meeting notes. One of our members has transcribed "minutes" during every meeting outlining the expected tasks for each person during the current sprint.

The advantages of using Google Drive are that we can modularize our project's documentation, organizing files to be within sub-directories. We also can edit the information that we have access to in real-time, meaning communication between us is instantaneous and there is no latency in the exchanging of information.

3.4. Inter-group Communication

3.4.1. Supergroup Communication

In order to make sure all groups could play online successfully with one another, the networking protocol has been revised multiple times to reduce ambiguity, and the discovery protocol we made has been tweaked to clarify it and further specify it.

In order to coordinate the groups within the Super Group, throughout the year we had meetings wherein a representative from each group meets with the other group representatives regularly to discuss and document changes.

The merge approval count on GitHub has been set at a minimum of 3 to ensure there is sufficient agreement for each change.

3.4.2. Standards for Bridge

Alongside the language we'd defined to represent games, we had also agreed as a supergroup on a series of de facto rules for implementing certain games.

In bridge, there are many variations to how the game can be played in terms of bid doubling and redoubling. The supergroup has agreed that the **only** opposition to the person making a bid (the "declarer") can double it's value.

We also elected that the bidding part of Bridge would end once three "passes" in a row occurred during the auction, i.e. that 3 players would pass the opportunity to play a bid after someone has made a bid. Therefore, we have programmed our clients such that in bridge, if the first three moves are a pass, the last player in the square **must** make a bid.

We also had to define our own scoring rules for Bridge as a super group, because there are many variations to the game and how it is scored online.

3.5. Setbacks and Plan Revision

3.5.1. COVID-19

As a result of the COVID-19 pandemic, alterations to our SCRUM had to be implemented. In short, we added an extra sprint to our plan and revised our requirements, focusing on creating a robust and functional product that met the requirements rather than expecting to be able to create a complicated program under changed, more stressful conditions.

Supergroup communication became much more difficult because even though we could call as a group of 5 using VOIP programs such as Messenger or Teams, the supergroup of 40+ people could not do a group call in that way for many reasons, both logistical (time zones, getting everyone online at once) and technical (no typical home network/social media platform would be able to support such a large group call). This made inter-group testing much more difficult, and resulted in us having to test a lot of games within our group.

In-person meetings could no longer happen because of social distancing rules and geographical distancing, so the typical groupwork sessions that we did before were no longer possible. This meant that we needed to keep in touch and ensure our documentation was as detailed and understandable as possible, to minimize misunderstanding and ensure the best could be made of our remote group programming situation. This was crucial even though it was more difficult than ever after the pandemic because we could not explain difficult concepts to each other as easily as we could in person.

In addition to this, one of our group members was being shielded and did not have good equipment at home to work to the best of her ability. This hindered our ability to work as a group.

We also lost quite a good deal of time setting up the program on our home machines, as this involved generating public keys, linking them to gitlab and getting the dependencies set up. We managed to help each other through this and got it working within a week, despite time zones and other issues.
We also could not access the gitlab website outside of the Computer Science network. We circumvented this by using nginx to proxy pass our computer science domain of one member(sjs31) to the Gitlab.

3.5.2. Microsoft Teams

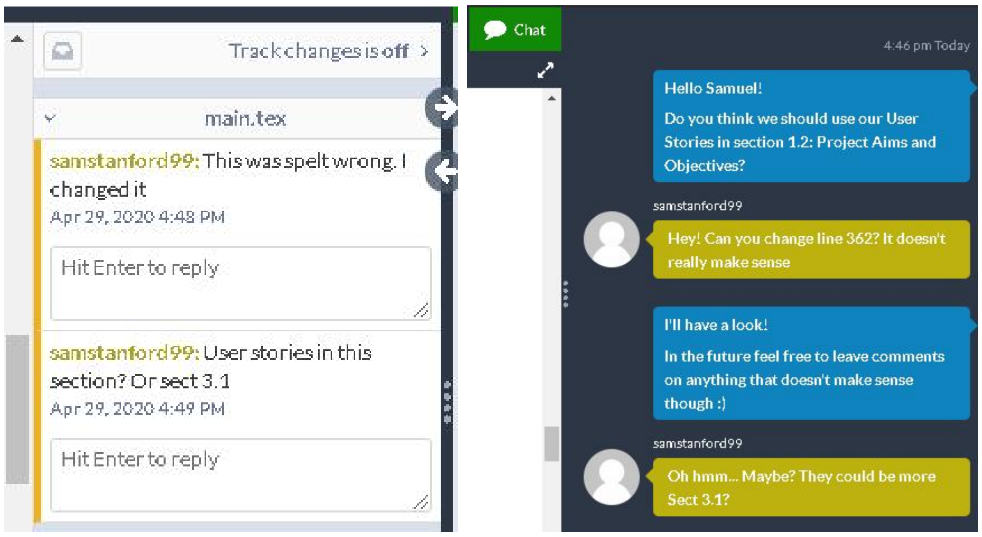
We used MS Teams for meetings and communication. We primarily used MS Teams for meetings with the group supervisor at the start. We had tried the video conference software zoom before but it didn't work for our devices as well additionally, some of us already using ms teams for some other modules.

We also used it for working together as it also had screen-share functionality. Unlike Facebook messenger, MS Teams had only been used for academia, making it great for the group to focus on work and get less distracted. As the group was checking it everyday, it helped us stay updated with each other's work.

3.5.3. Overleaf

For all our reports we used the web-based writing software Overleaf to work collaboratively. We used it because of its ability to allow multiple people to write and edit the document at the same time. It also has a chat function to aid this.

After COVID-19, it was useful as we could no longer meetup in the labs to work on sections together. Instead, we scheduled sessions based on time zones where we would call and work on the report together. It was an advantage that it was web-based as, we may have different systems and software at home that may not be compatible.


Track changes is off

main.tex

samstanford99: This was spelt wrong. I changed it
Apr 29, 2020 4:48 PM

Hit Enter to reply

samstanford99: User stories in this section? Or sect 3.1
Apr 29, 2020 4:49 PM

Hit Enter to reply

Chat

4:46 pm Today

Hello Samuel!

Do you think we should use our User Stories in section 1.2: Project Aims and Objectives?

samstanford99

Hey! Can you change line 362? It doesn't really make sense

I'll have a look!

In the future feel free to leave comments on anything that doesn't make sense though :)

samstanford99

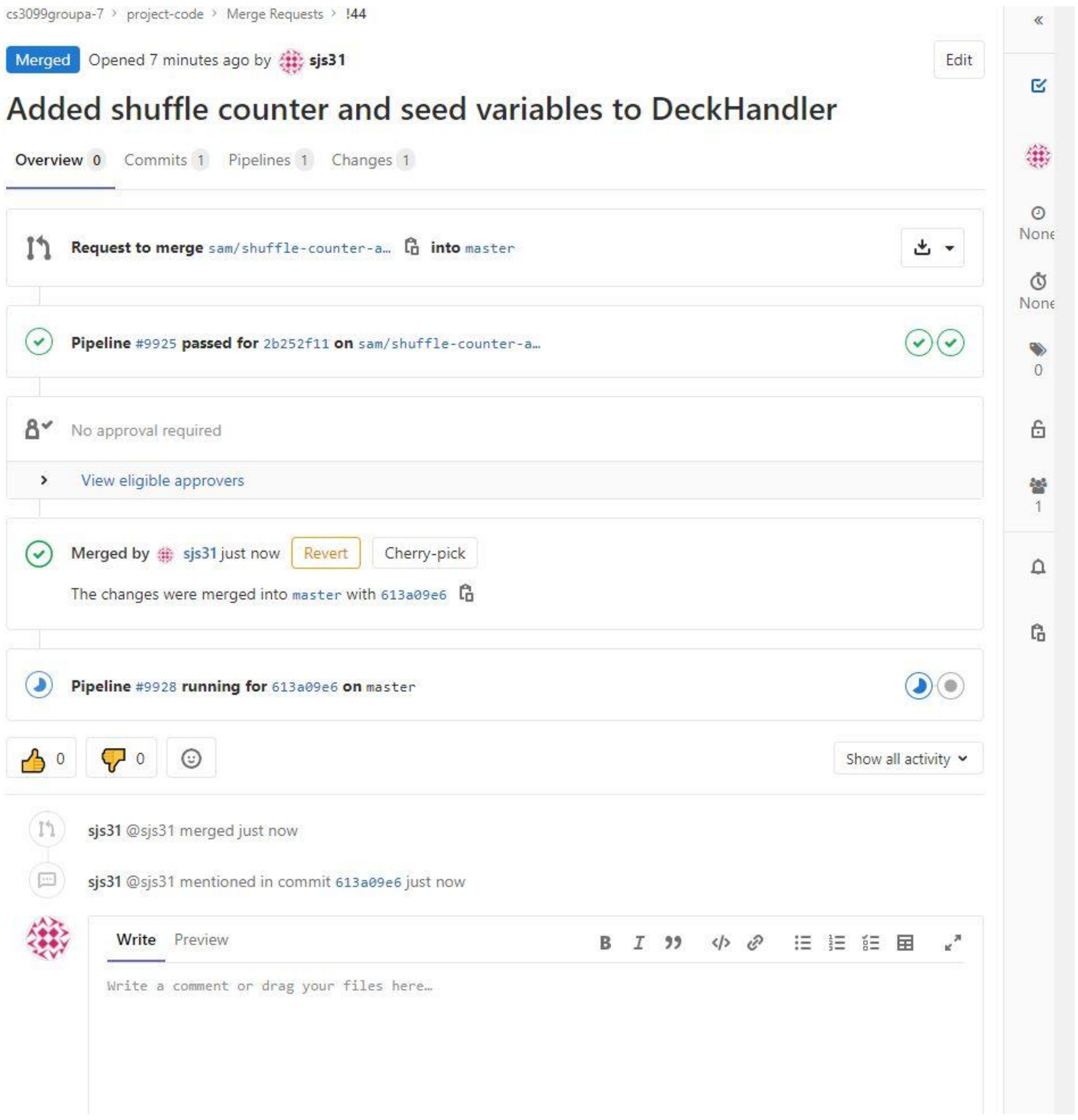
Oh hmm... Maybe? They could be more Sect 3.1?

Figure 3.5.: The collaborative features of Overleaf.

3.6. Lessons Learned and Future Changes

4. Testing

4.0.1. Pipeline



cs3099groupa-7 › project-code › Merge Requests › !44

Merged Opened 7 minutes ago by sjs31

Added shuffle counter and seed variables to DeckHandler

Request to merge sam/shuffle-counter-a... into master

Pipeline #9925 passed for 2b252f11 on sam/shuffle-counter-a...

No approval required

View eligible approvers

Merged by sjs31 just now

The changes were merged into master with 613a09e6

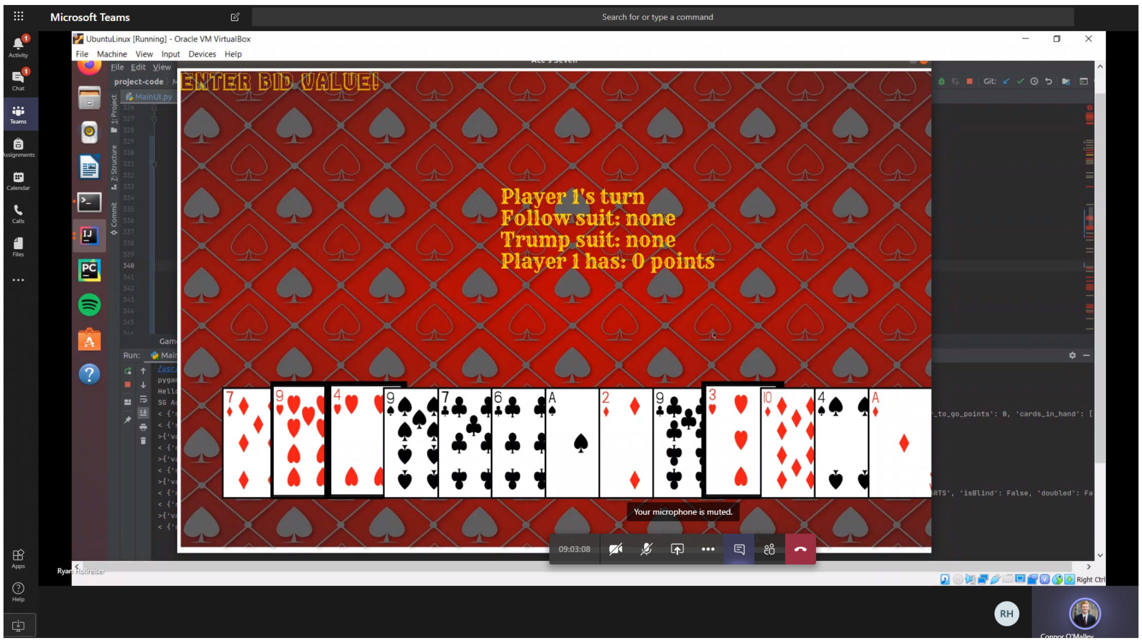
Pipeline #9928 running for 613a09e6 on master

sjs31 @sjs31 merged just now

sjs31 @sjs31 mentioned in commit 613a09e6 just now

Write a comment or drag your files here...

Figure 4.1.


ENTER BID VALUE!

Player 1's turn
Follow suit: none
Trump suit: none
Player 1 has: 0 points

Microsoft Teams
Search for or type a command
UbuntuLinux [Running] - Oracle VM VirtualBox
Your microphone is muted.

Figure 4.2.: Our process of collaboratively linking the front and back ends during an MS teams call. Here we can see the system status displayed on the screen as well.

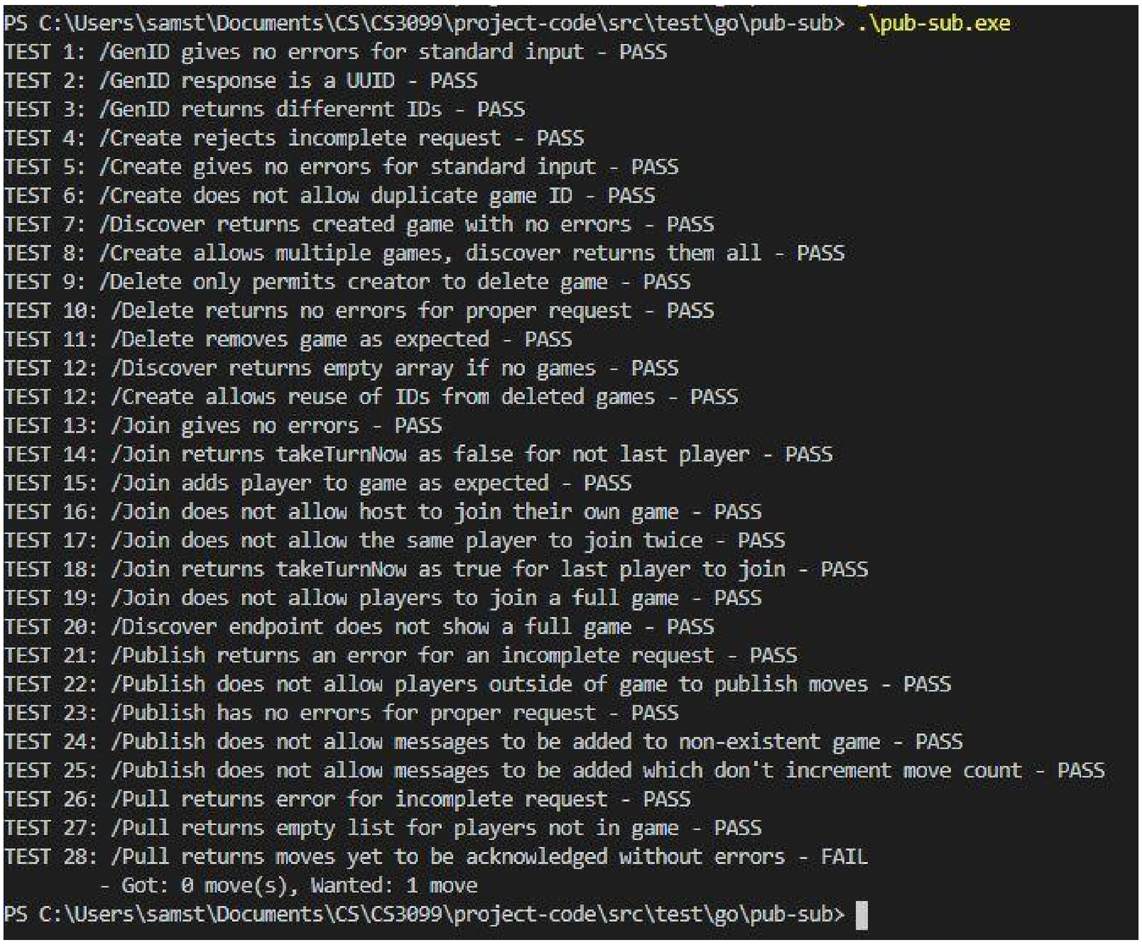
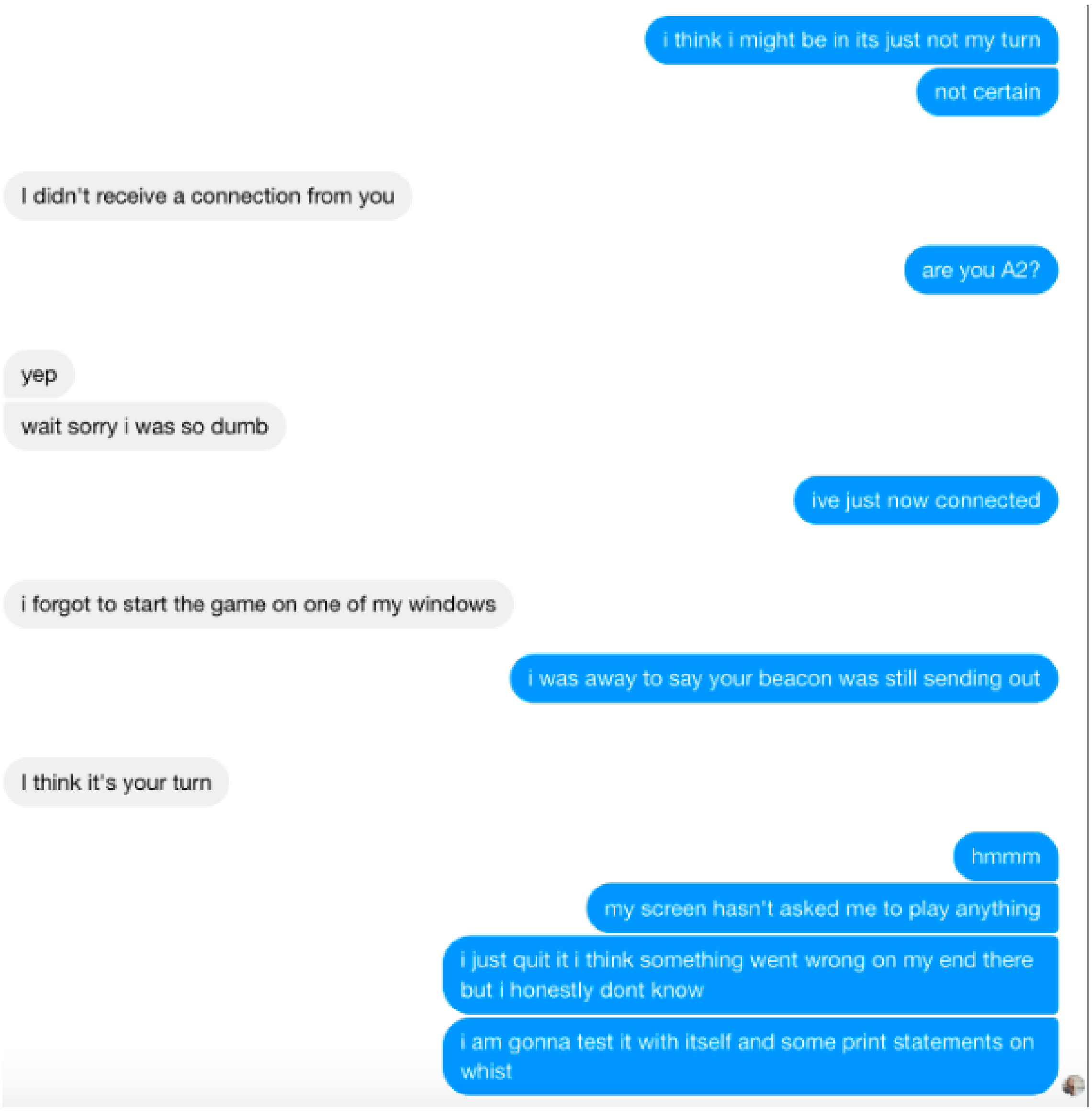

```
PS C:\Users\samst\Documents\CS\CS3099\project-code\src\test\go\pub-sub> .\pub-sub.exe
TEST 1: /GenID gives no errors for standard input - PASS
TEST 2: /GenID response is a UUID - PASS
TEST 3: /GenID returns differernt IDs - PASS
TEST 4: /Create rejects incomplete request - PASS
TEST 5: /Create gives no errors for standard input - PASS
TEST 6: /Create does not allow duplicate game ID - PASS
TEST 7: /Discover returns created game with no errors - PASS
TEST 8: /Create allows multiple games, discover returns them all - PASS
TEST 9: /Delete only permits creator to delete game - PASS
TEST 10: /Delete returns no errors for proper request - PASS
TEST 11: /Delete removes game as expected - PASS
TEST 12: /Discover returns empty array if no games - PASS
TEST 12: /Create allows reuse of IDs from deleted games - PASS
TEST 13: /Join gives no errors - PASS
TEST 14: /Join returns takeTurnNow as false for not last player - PASS
TEST 15: /Join adds player to game as expected - PASS
TEST 16: /Join does not allow host to join their own game - PASS
TEST 17: /Join does not allow the same player to join twice - PASS
TEST 18: /Join returns takeTurnNow as true for last player to join - PASS
TEST 19: /Join does not allow players to join a full game - PASS
TEST 20: /Discover endpoint does not show a full game - PASS
TEST 21: /Publish returns an error for an incomplete request - PASS
TEST 22: /Publish does not allow players outside of game to publish moves - PASS
TEST 23: /Publish has no errors for proper request - PASS
TEST 24: /Publish does not allow messages to be added to non-existent game - PASS
TEST 25: /Publish does not allow messages to be added which don't increment move count - PASS
TEST 26: /Pull returns error for incomplete request - PASS
TEST 27: /Pull returns empty list for players not in game - PASS
TEST 28: /Pull returns moves yet to be acknowledged without errors - FAIL
        - Got: 0 move(s), Wanted: 1 move
PS C:\Users\samst\Documents\CS\CS3099\project-code\src\test\go\pub-sub>
```

Figure 4.3.: Our 28 tests for the pub/sub system, all passing.

4.0.2. Piping (UX to back-end)

4.0.3. Publisher/Subscriber System (Pub-Sub)

4.1. Inter-group Testing



i think i might be in its just not my turn

not certain

I didn't receive a connection from you

are you A2?

yep

wait sorry i was so dumb

ive just now connected

i forgot to start the game on one of my windows

i was away to say your beacon was still sending out

I think it's your turn

hmmm

my screen hasn't asked me to play anything

i just quit it i think something went wrong on my end there but i honestly dont know

i am gonna test it with itself and some print statements on whist

Figure 4.4.: An exchange we had with another team from the supergroup during testing.

5. Evaluation

5.1. Against Project Criteria

We can say with certainty that we achieved each of the project criteria to a certain degree. How ever the criteria was not fulfilled as we would like.
The functionality of the card game bridge across supergroup was hindered by our inability to work as effectively across distance across distances due to the pandemic and therefore bridge works with limited functionality across the supergroup.
The AI for the project is not smart, learning AI that adapts to different card games but it is functional and does play valid moves.
the UX was surposed to

5.2. Against Team-defined Goals

A. SCRUM Plans/Stand-Up Meetings

A.1. November 2019

A.2. January 2020

A.3. March 2020: COVID-19 Plan

A.4. All Stand-Up Meetings

B. User Stories

B.1. November 2019

B.2. January 2020

C. Weekly Progress Reports

C.1. Semester 1

C.2. Semester 2

D. Misc.

D.1. List of Playable Games

1. Bridge
2. Catch the Ten
3. Clubs
4. Contract Whist
5. Jabberwocky
6. No Pass Hearts
7. Oh hell
8. One trick pony
9. Reverse Spades
10. Smart Aleck
11. Simplified Napoleon
12. Spades
13. Whist
14. Speed Golf Whist
15. Speed Whist
16. Two Player Whist

E. References/Sources

Bultel, Xavier, and Pascal Lafourcade. “Financial Cryptography and Data Security Conference.” HAL, 19 Dec. 2018, hal.archives-ouvertes.fr/hal-01959762/document. HAL Id: hal-01959762

Gothelf, Jeff, and Josh Seiden. Lean UX: Designing Great Products with Agile Teams. O’Reilly, 2019.