Exploding Kittens: Client Package

Nathaniel Blanco, Dalton Julian,
Braxton Lee, Andrew Orians

Team 5

May 13, 2022

Table of Contents

Introduction

Exploding Kittens is a multiplayer friend vs. friend card game, where the goal is to be the last one standing. In this digital version, up to 10 players can join the party. This is a Java software project designed to run on a PC or laptop, providing a play window for each player.

This document is intended to provide the user with all necessary information on the software. First, the user manual provides a clear walkthrough of all actions that can be taken in the game. Instructions for installing the software can be found next under the installation guide. The development and maintenance guide contains information on tools and libraries used, as well as how to maintain and troubleshoot the system. The software requirements specification notes where the rules of Exploding Kittens can be found, along with all major system requirements. The software architecture section gives a flyover view of how the classes of the system are designed and how they interact. Finally, the test plan provides an overview of the testing strategy carried out by the development team, as well as a test summary that details the specific unit, integration, system, and acceptance tests, both manual and automatic, that are performed on the system.

User Manual

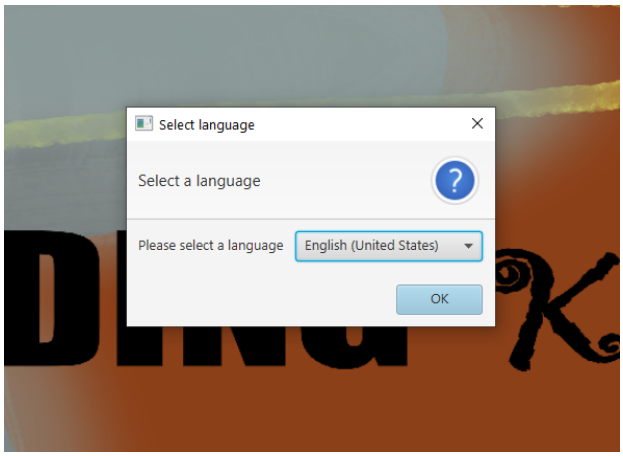
This user manual will help guide first-time users through the process of using the game, with step-by-step instructions and accompanying images.

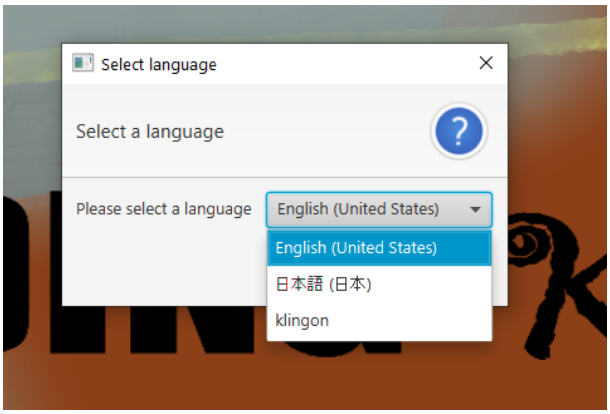
Game Setup

Start menu

Upon starting the game, the start menu provides two options: Start Game on the left, and Change Card Images on the right. Selecting the left button begins setup for a game of Exploding Kittens, and brings up a pop-up to change the language.

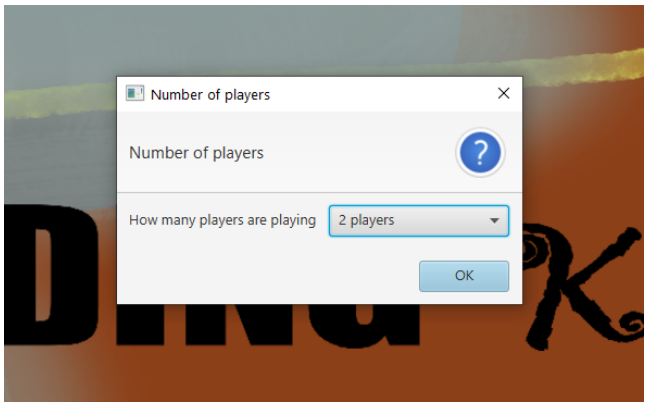
Changing the language

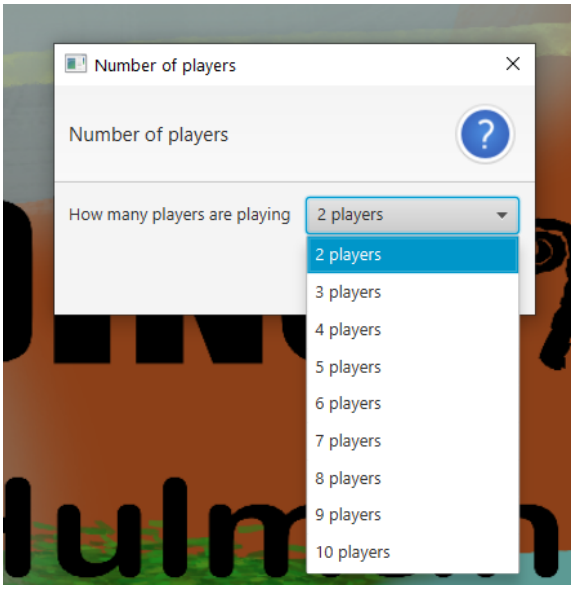




Using the drop-down, one can select from available language settings to match their preferences, then select OK to continue. A new pop-up will appear to ask for a number of players.

Selecting the number of players



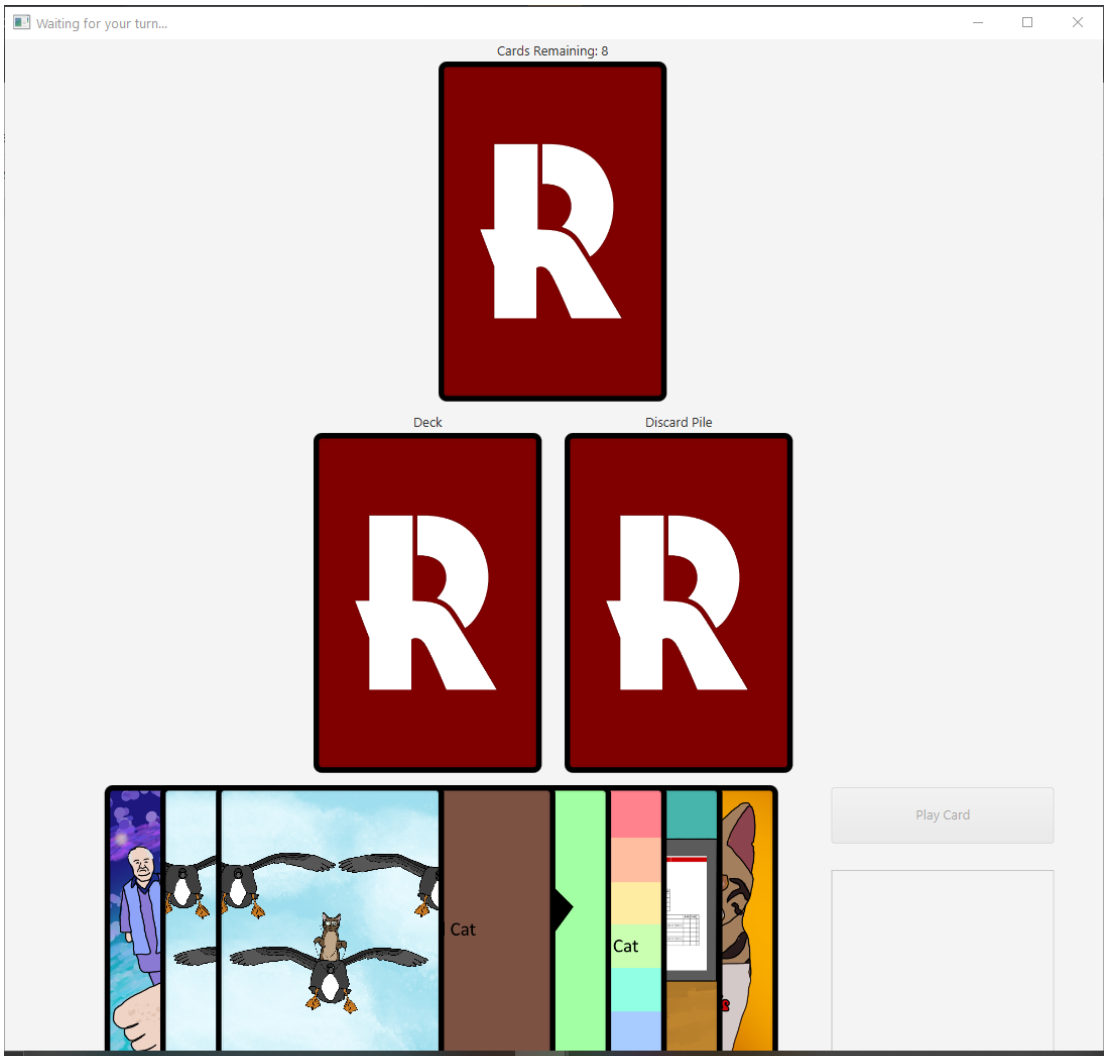


The drop-down on the new pop-up allows users to select how many players will be playing. The game supports a range of players from 2 to 10. Upon selecting a number and pressing OK, the game will generate a play window for each player, along with a dialog box for each player, asking for entry of the player's name.

Enter each player's name

For each player, one of these windows will appear, into which a name can be entered for that player. The windows are not specific to any one player, so player order is intended to be random depending on who enters their name into which dialog.

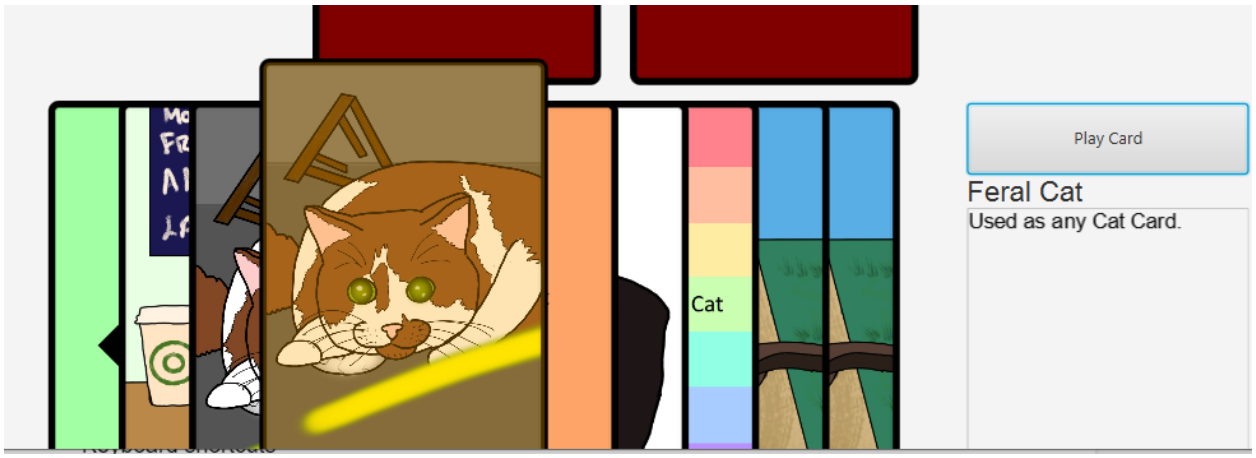
Handling the player windows



A play window is generated for each player while their name is being entered. The title of the window, in the top left, indicates whether or not it is that player's turn. After each turn is

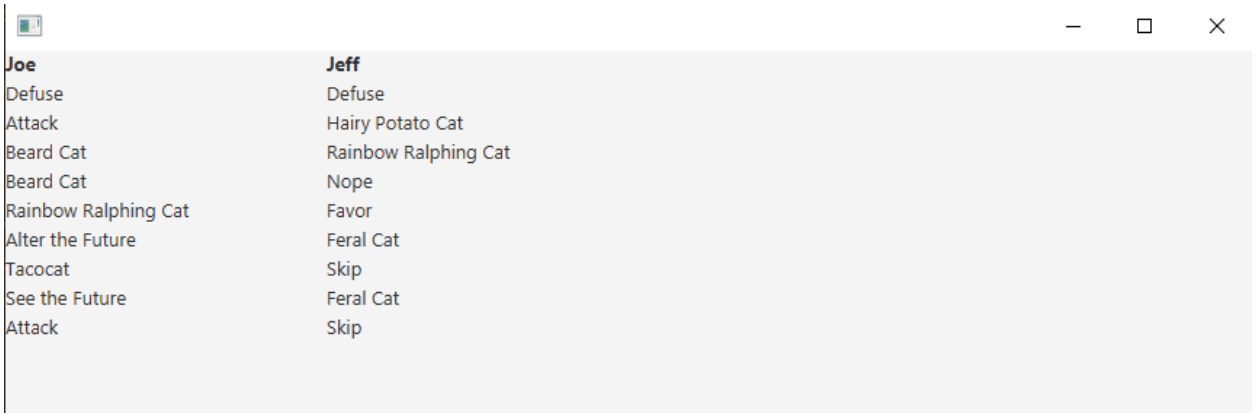
completed, the game will bring the next player's window to the top, allowing the device running the game to be handed off in a pass-and-play style. Alternatively, each window can be moved to additional monitors or placed side-by-side to allow each player to view their hand at the same time.

At the top are the decks of the opponent players, along with how many cards each player has left. The Rose-Hulman logo indicates the backside of a card. Below that are two stacks of cards, labeled Deck and Discard Pile. The deck can be clicked to draw from it when it is that player's turn, and any played cards will visually move to the discard pile. At the bottom of the window is the player's hand. Hovering the mouse over each card brings it to the front to view.



Selecting a card by clicking on it brings up a description of the card on the right, and enables the "Play Card" button for playing the selected card.

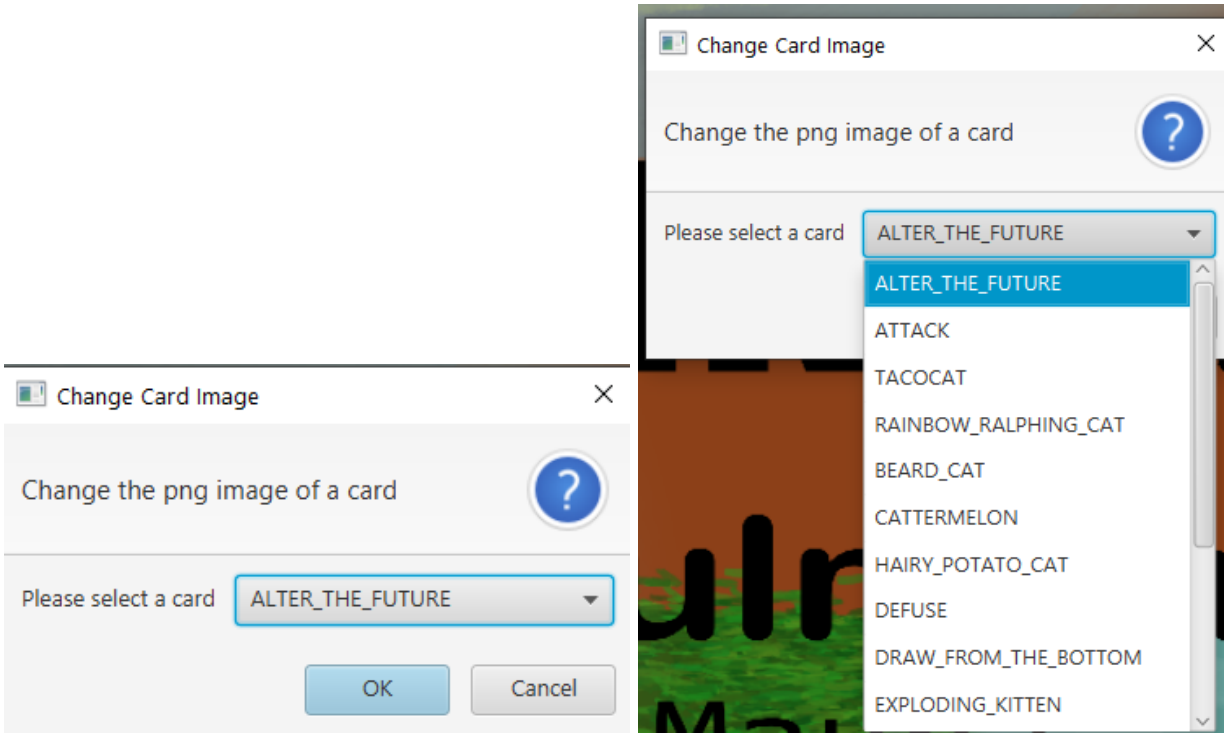
The spectator view



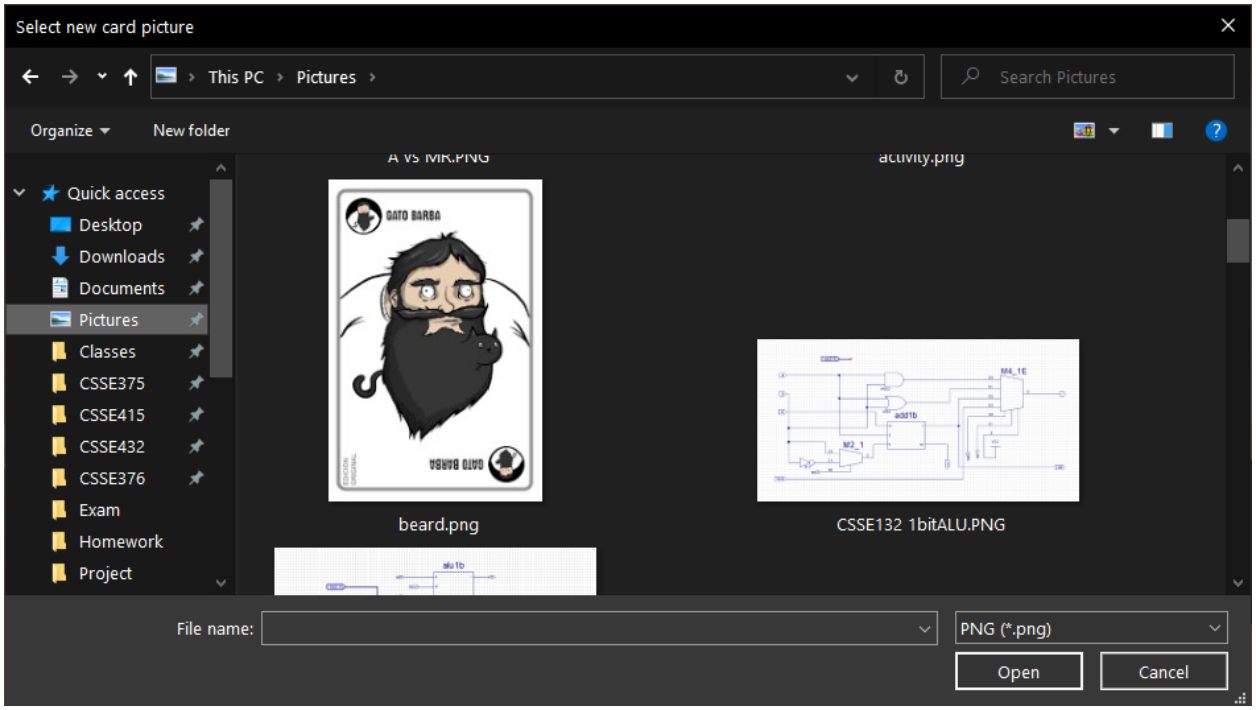
Along with the player windows, a spectator view window will be created, which lists each player along with all of the cards that are currently in their hand. This can be moved to display somewhere away from the game windows, allowing anyone who is not playing to get an inside look into how the game is playing out.

Changing the card's images

On the home screen, instead of starting the game, the Change Card Images button can be selected to customize the art of each card. A dialog will appear to allow you to select which card to change.



After selecting a card from the drop-down and pressing OK, the user will be prompted with a file selector.

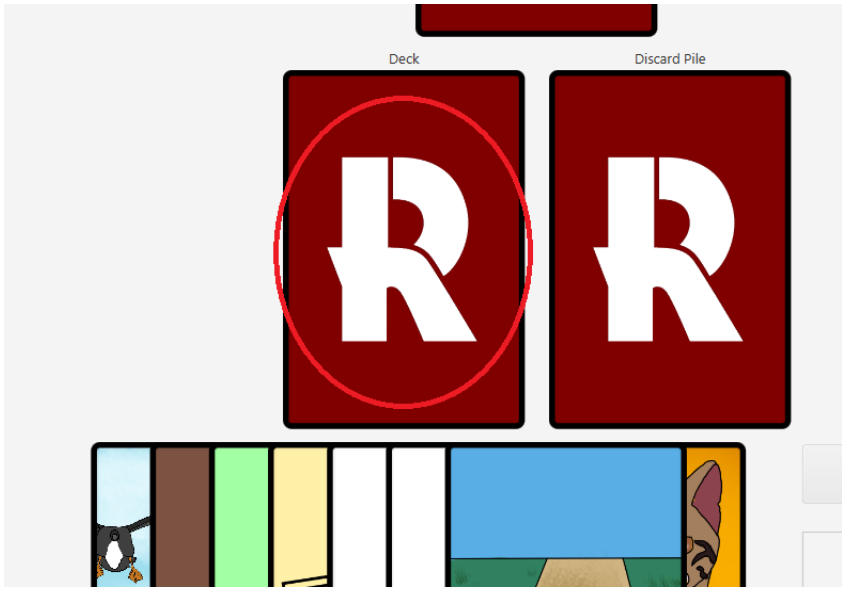


A .png file can be selected from the user's computer. After selecting the image and then Open, the card art for that particular card will be replaced with the user's selection, and the user will be returned to the main menu.

Game Play

Basic actions

Draw a card

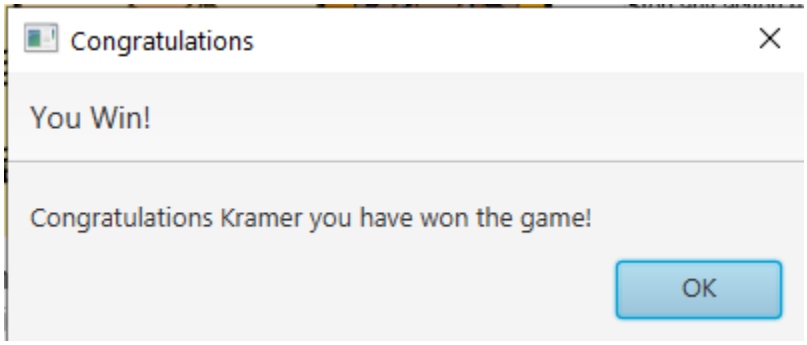


When it is time for a player to draw a card, this can be done by clicking on the deck pile. A new card will move from the pile to the player's hand.

Select a card and play

To play a card, one can simply click on that card in their hand at the bottom of their play window. In order to play the card, click the Play Card button. Different cards have different effects that often generate pop-ups to control. See below in the "Special Card Effects" section for details on how to play each type of card.

Winning the Game



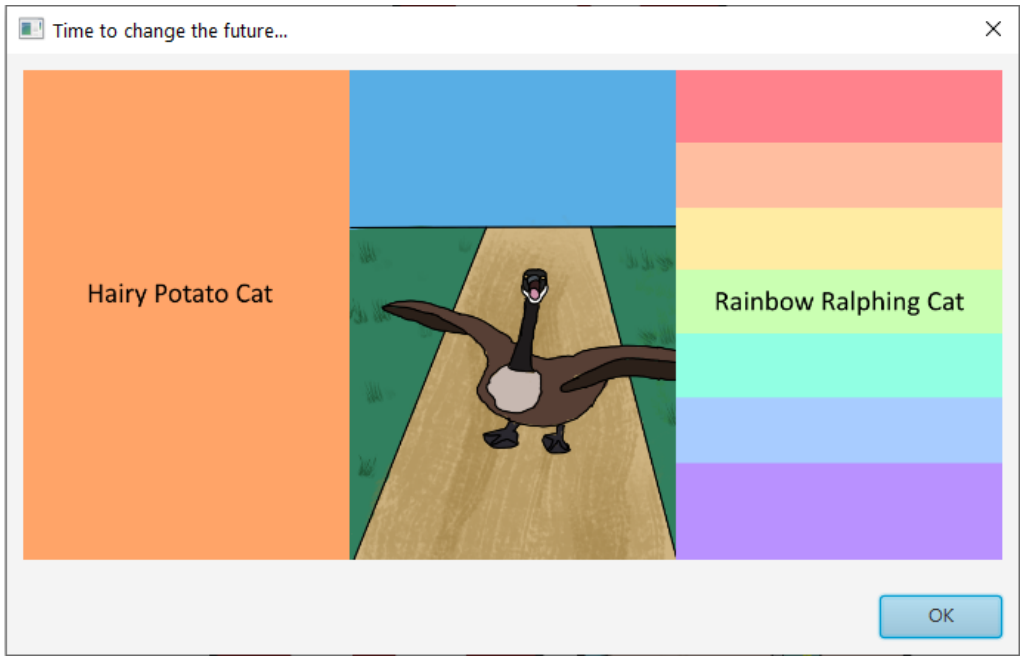
When the second-to-last player draws an Exploding Kitten with no defuse card, they'll be knocked out, and the final player will be congratulated with a message to let them know that they are the winner of the game.

Keyboard shortcuts

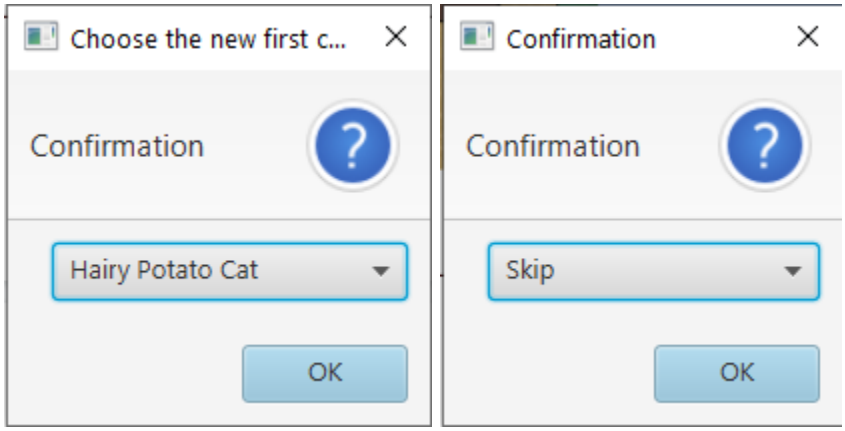
- The row of numbers on the keyboard 0 through 9 can be pressed to select/deselect a card in the hand
- SHIFT can be used to draw a card
- ENTER / SPACE can be used to play a selected card

Special Card Effects

Alter The Future

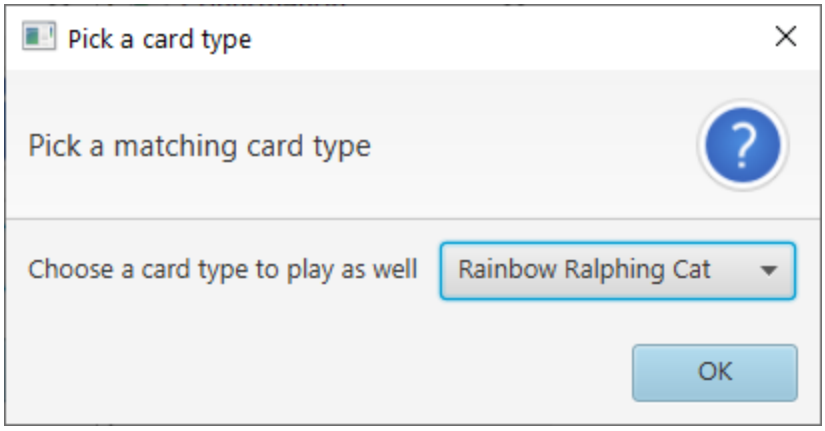


When a player plays an Alter the Future card, a dialog will appear, displaying the three upcoming cards from the deck in order. After looking at this and selecting OK, the player will be prompted to reorder these.



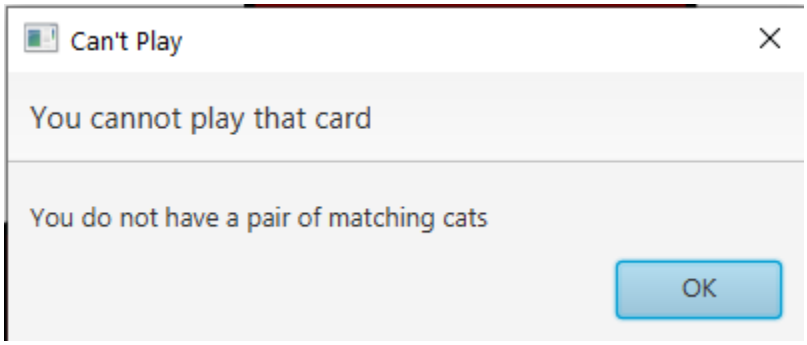
The first pop-up asks for the new first card (essentially, the card that will be on top of the deck afterward). The drop-down is used to select which of three cards is on top. Next, a second dialog will display to ask which card will be second. After selecting both of these, the order of the deck will be altered in the way chosen by the user.

Cat Cards



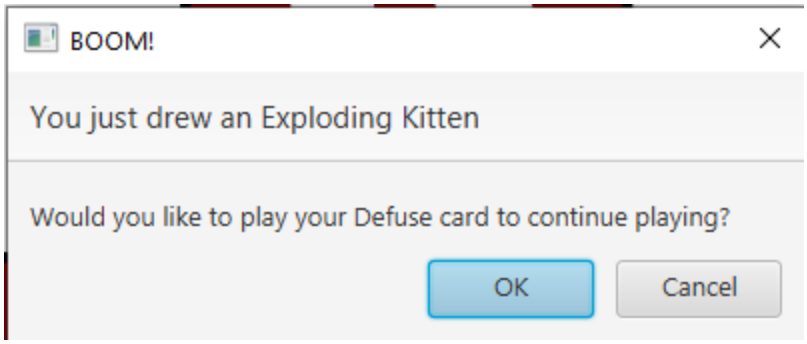
When ‘Play Card’ is clicked while selecting a Cat Card, the card will not immediately be played. A pop-up will appear, asking the user to select which card they wish to play it with. In order to play the Cat Card, a matching card must be available in the player’s hand, which must then be chosen using the above pop-up. Pressing OK after selecting the match from the drop-down will proceed to play both cards.

After playing a pair of Cat Cards, a new pop-up will appear, prompting the player to select an opponent to steal a random card from. The drop-down can be used to select an opponent.

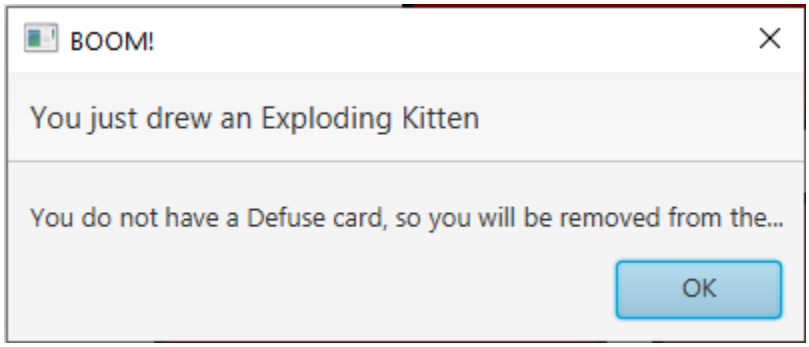


If the player has no matching Cat Cards that can be played with the first card they attempted to play, a dialog will appear to notify the player that they cannot play that card without a match.

Exploding Kitten

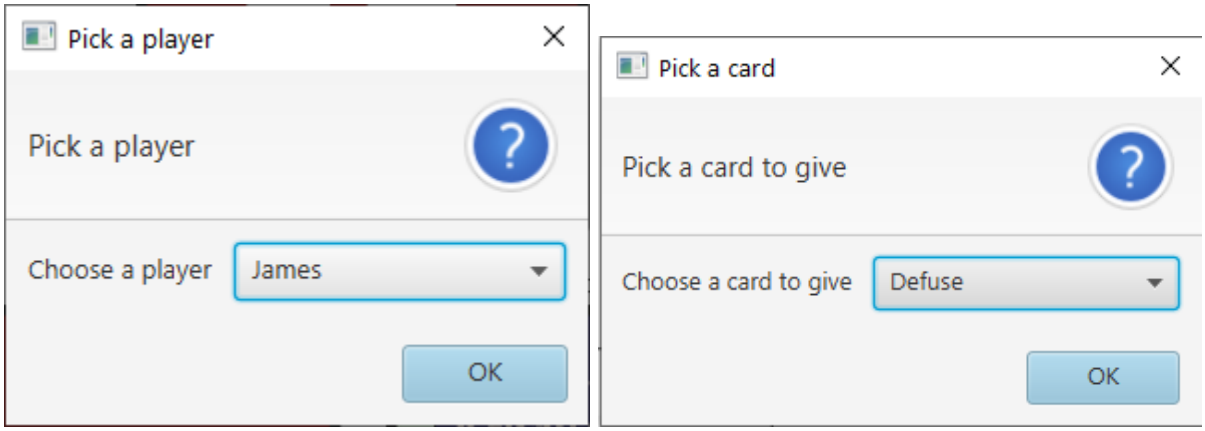


If a player draws an Exploding Kitten with a Defuse card in-hand, the player is prompted with asking if they would like to use their Defuse card to avoid the Exploding Kitten and continue playing. Pressing OK will use the Defuse. Choosing Cancel causes the player to lose, and be removed from the game.



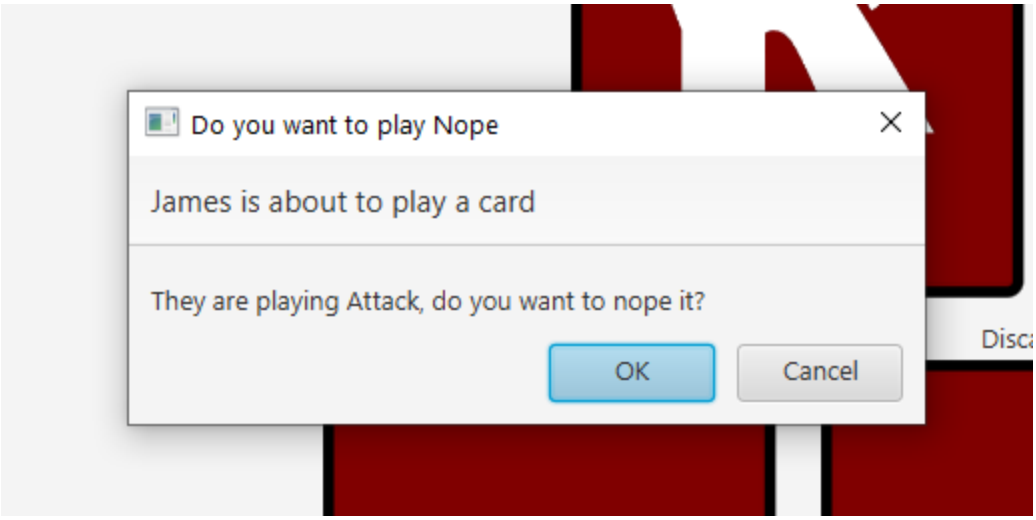
If a player draws an Exploding Kitten card with no Defuse in-hand, they're out of the game. A pop-up will appear, letting that player know that they've been removed from the game.

Favor



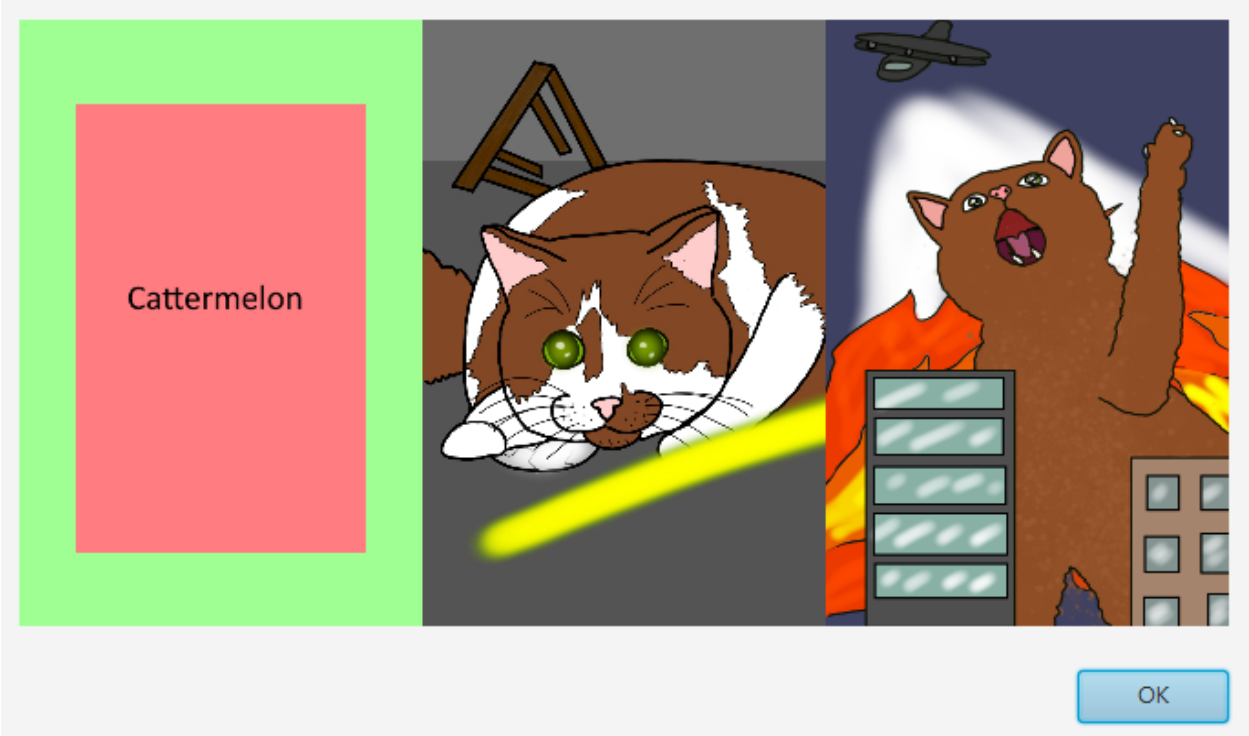
After playing a Favor card, a player can select which opponent they would like to take a card from using the drop-down from the 'Pick a player' pop-up. The game will then bring up another pop-up for the targeted player to use, in which they must use the drop-down to select which card they are going to give up.

Nope



Whenever a card with a special effect is played, any of the other players may have a Nope card that can be triggered. To “nope” the action of the card that an opponent has played, a user can select OK. Selecting Cancel allows the original player’s card to play out.

See The Future



After playing a See the Future card, the user will be presented with a pop-up that displays the next three cards in order. In other words, in the image above, Cattermelon is on top of the deck, followed by Feral Cat and the Exploding Kitten. Selecting OK closes the view.

Developer Installation Guide

Our project is set up using Gradle, so installation and setup are fairly straightforward. Simply build the project using Gradle, and it should take care of installing and managing all the dependencies. The combination of libraries we use requires Java 11 or later and SDK version 50 or later. The process for this varies by IDE and is well-documented, so it will not be explained here.

If running from an IDE, we have found that IntelliJ is most preferable. There are slight differences in how various IDEs handle file paths, so if you wish to run from a different one, they will have to be modified. The three major instances are FXML_BASE_PATH in FXMLLoader, CARD_BACK_PATH in UiCard, and BASE_PATH in CardType. Depending on the IDE, you may or may not need to remove the leading “app” from the path. We have taken efforts to ensure that these difficulties appear in as few places as possible.

Development and Maintenance Guide

Recommendations and Best Practice

This project was originally developed using test-driven development, and we recommend that any future developers do the same. This ensures that all code is comprehensively tested and its behavior is well-defined before it is written.

We also recommend that feature and change management follow industry standards. We have created a board on Trello to track the status of various features and manage which developers are working on which ones. We developed using a branch for each new feature (or other set of work), and then only merged our changes when finished. We also kept a separate branch for each milestone of the project, although this could also be accomplished using releases at specific points in time. We recommend that future developers perform this separation in a similar way.

IDE

Due to slight differences in the ways various IDEs handle file paths, we advise all developers using the same IDE (or ones that handle paths in the same way, if they do not wish to change their settings for this). This project was developed using IntelliJ, and we recommend this for future developers, as well. Further instructions can be found in the installation guide.

Operating System

In order to get the JAR to compile properly, we needed to make some of the file paths we specified specific to Windows systems. We cannot guarantee they will run on other operating systems. Most of these are commented out in order to preserve functionality for other features, but this is something to keep in mind.

Compatibility

Libraries

JavaFX

We used the JavaFX library for our graphics and UI management. This provides some level of abstraction to make graphics easier to handle and understand. The documentation for it can be found here. Unfortunately, the way this framework behaves, it makes testing fairly unique.

Firstly, instead of having each test method stand on its own, we must declare them like any other method and call them when the test class is initialized to ensure everything is in the proper context. Also, because of the way this test environment works, continuous integration is unable to run these tests, since it cannot create any of the necessary UI elements. For this reason, many of our UI tests are commented out as a quick fix to get CI to run.

Other

In addition to JavaFX, our project also depends on several other libraries. However, these all serve to aid in the development process, and do not directly affect the functionality of the software. We use Checkstyle as a linter to ensure common style. We have Pitest set up for code coverage and mutation testing to help show the comprehensiveness and accuracy of our test suite. Spotbugs is a plugin which checks for common sources of bugs to catch them before they occur.

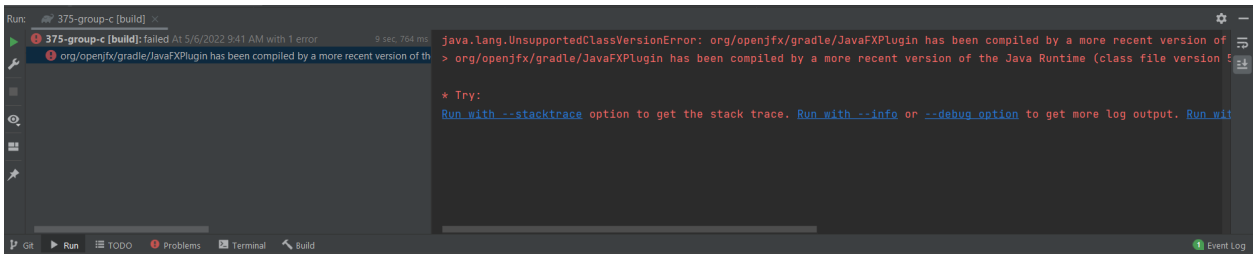
Continuous Integration

We have our continuous integration setup to simply build the project with Gradle. This ensures that not only is the project itself built and checked for errors, but any additional Gradle tasks are run as well. This means that all of our other checks are run, including the linter, code coverage, mutation testing, and Spotbugs. If any of these fail, it will be indicated to us, so we are instantly able to check not only the status and runnability of our code, but also its accordance with our standards for style and best practice. This CI action runs every time a push or pull request is made on the main branch in Github, and will prevent merging branches while there are failures.

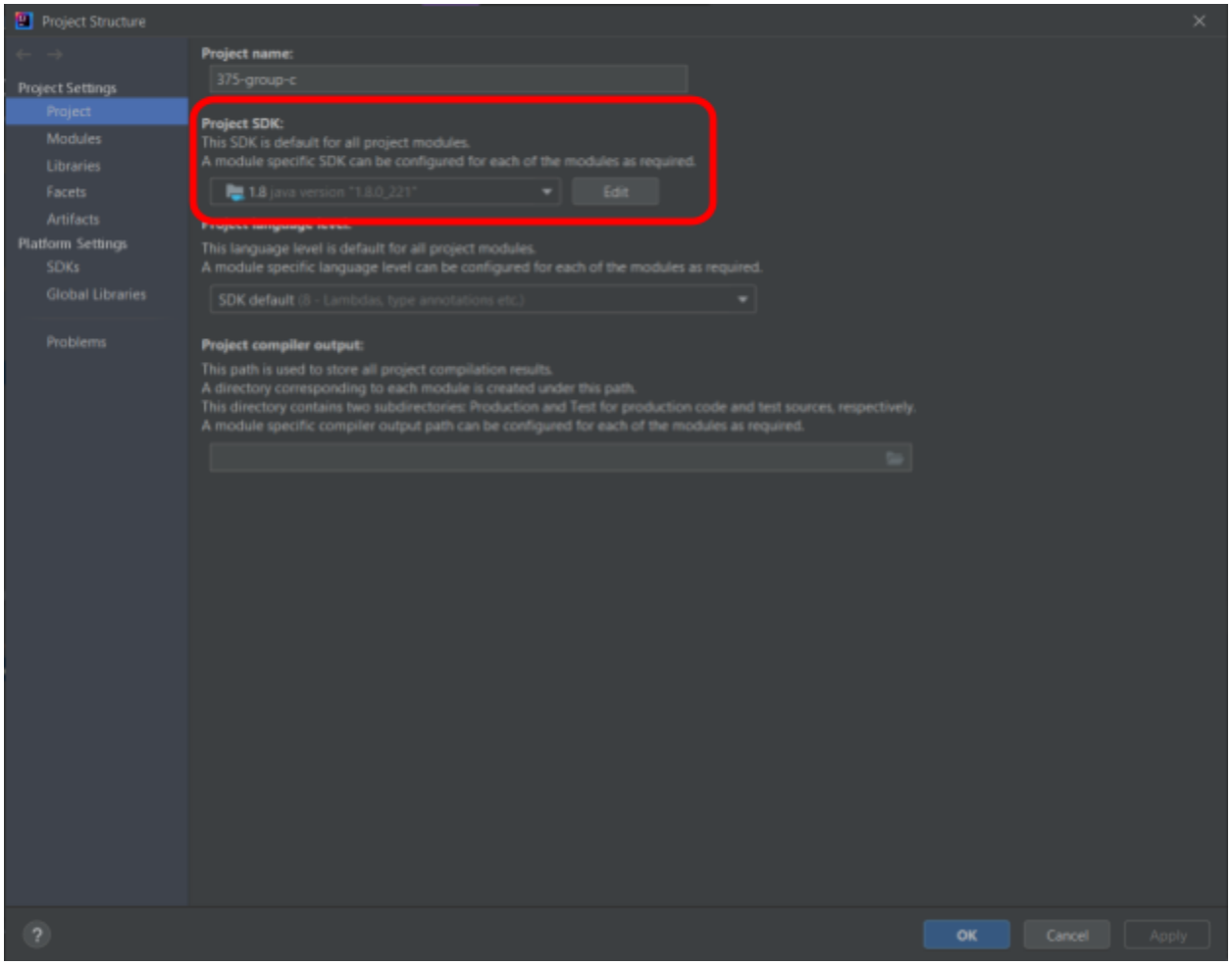
Troubleshooting

Versioning

The plugins used here require using Java 11 or greater. If this is not the case, the following error will appear when attempting to build the project:

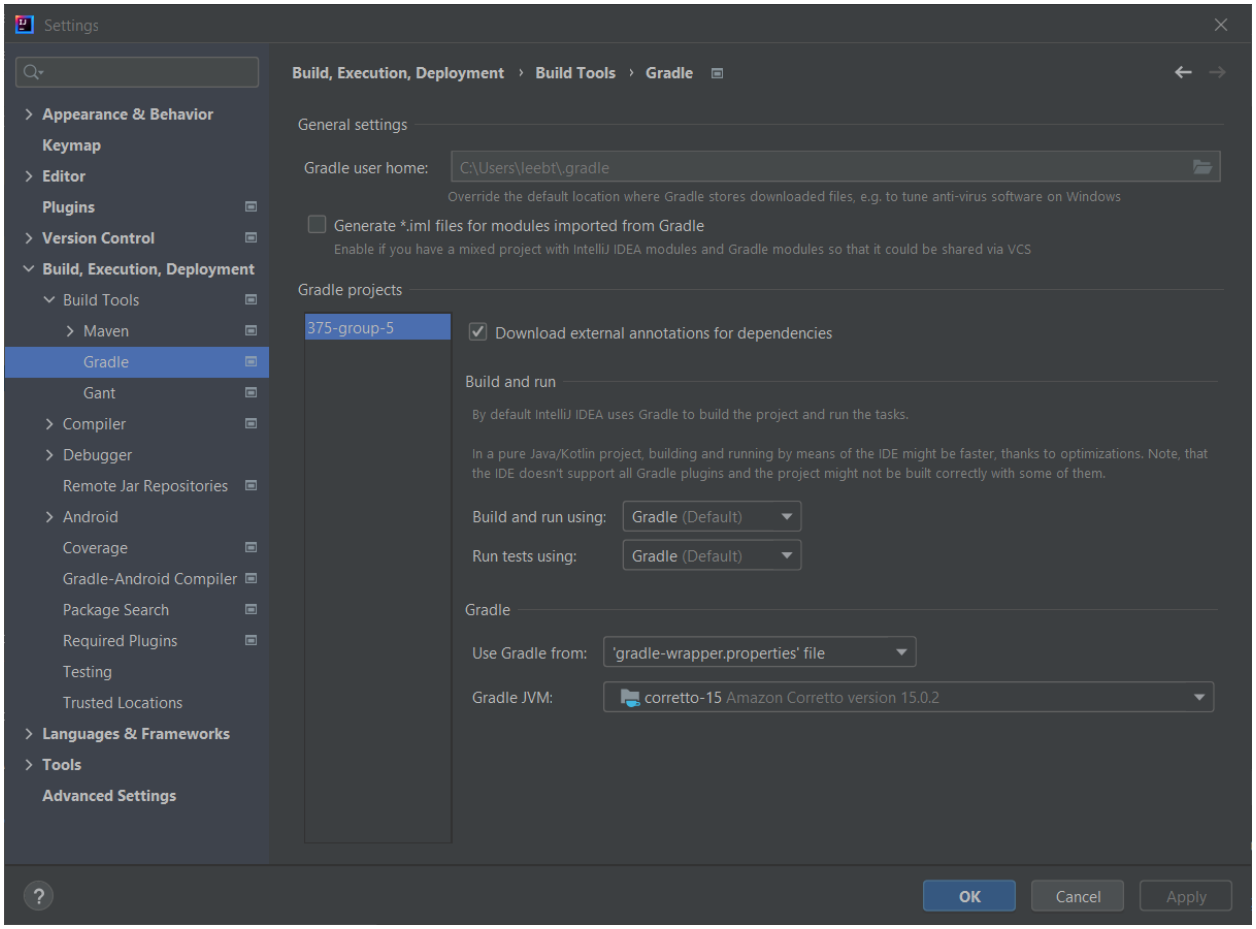


This can be fixed by changing the project to use a more recent version of Java, which in IntelliJ can be done by going to File > Project Structure > Project Settings > Project > Project SDK:



This may also be caused by the project version being correct, but the Gradle version being incorrect. It is possible to have a project run on, for example, Java 11, while Gradle builds its artifacts using Java 8. If this first solution does not work, then this can be checked and rectified

by going to File > Settings > Build, Execution, Deployment > Build Tools > Gradle > Gradle JVM:



JAR

As with the development version, the JAR must be run using the correct version of Java, which is 11 or greater. Attempting to run it by clicking on it will simply do nothing, but running it at the command line will produce the same error (talking about class file versions and being compiled with a different version of the Java Runtime). To rectify this, you will need to install a version of Java at least 11, which can be found on Oracle's website here.

Additionally, getting the JAR to recognize the proper files required some tampering with file paths. Because of this, they had to be made specific to Windows file systems, so we cannot guarantee the JAR will run on MacOS or Linux systems. Since all of our developers are Windows users, we unfortunately cannot provide any further testing or guidance.

Because JARs are designed to be archive files, you are not permitted to modify any of the files they contain. Unfortunately, this means we were unable to support custom card art in that version. Any attempts to do so will either result in errors, or simply fail. This is supported ONLY in the developer version, accessible from our GitHub repository. The code and paths for both

are present, but the JAR versions are commented out with notes. They are located in UiCard, CardType, and FXMLLoader. Additionally, one test each for Card and CardType are reliant on the path, so these are contingent on which version is being used. Currently, they are setup to support changing card art.

```
// For use with jar-compatible I/O
// private static final String BASE_PATH = "images/cards/";
private static final String BASE_PATH = "." + File.separator
        + "app" + File.separator + "src" + File.separator + "main"
        + File.separator + "resources" + File.separator + "images"
        + File.separator + "cards" + File.separator;
```

Possible Future Work

There were several features which we wished to add, but were unable to find the time or resources to complete. These can be exercises left for future developers. Some of these features include:

- Dynamic resizing or the window (currently, there is a bug in which the player views are slightly too tall for the screen)
- Support for custom images in the JAR version
- More types of cards
- The ability for players to choose subsets of cards to play with
- Custom player avatars
- Sound effects
- Characterization tests for UI (for the original project, automatic UI testing was not required)

Software Requirements Specification

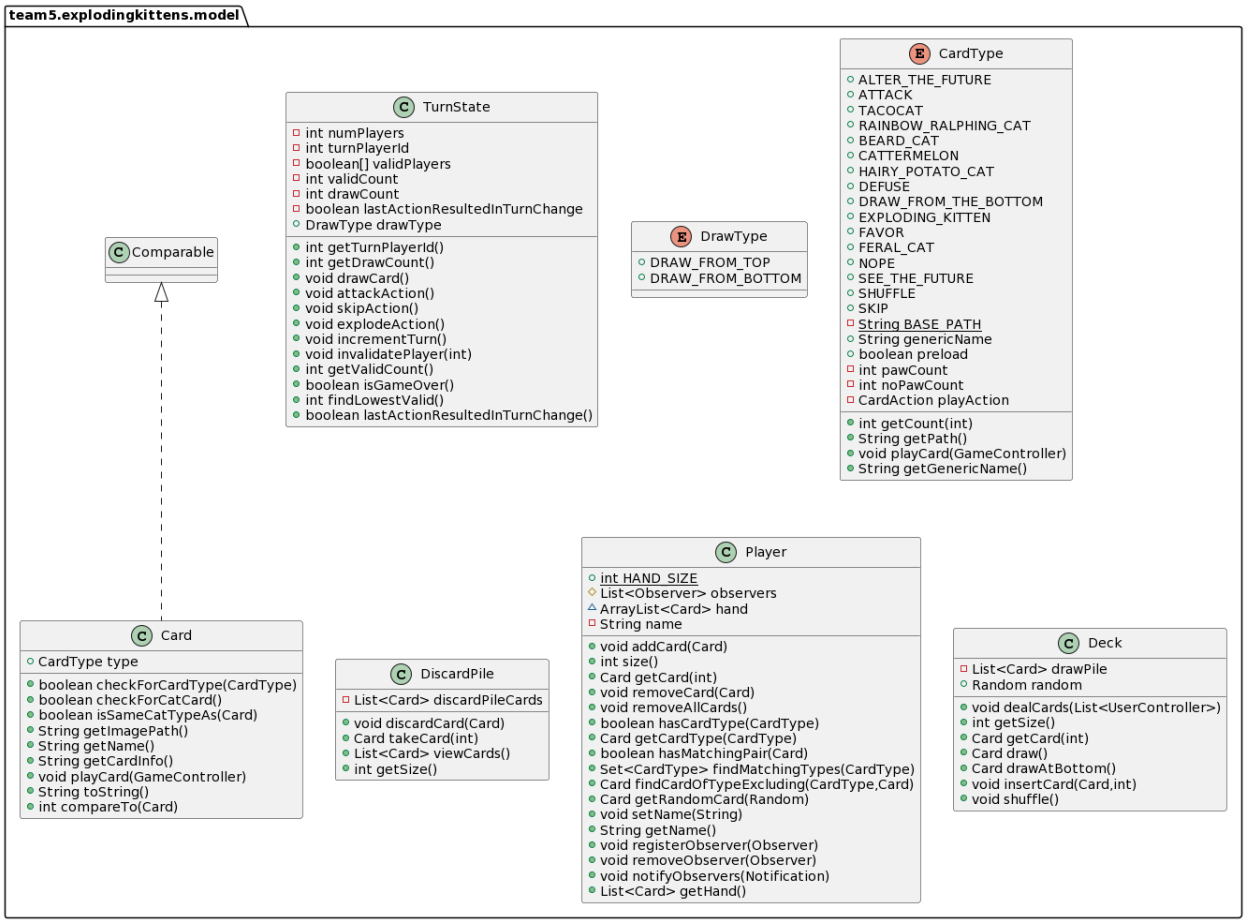
Our software will simulate the majority of elements of gameplay from the popular card game Exploding Kittens, and its first expansion pack.
The rules can be found here: https://www.explodingkittens.com/pages/rules-kittens-party
A simplified feature list is shown below.

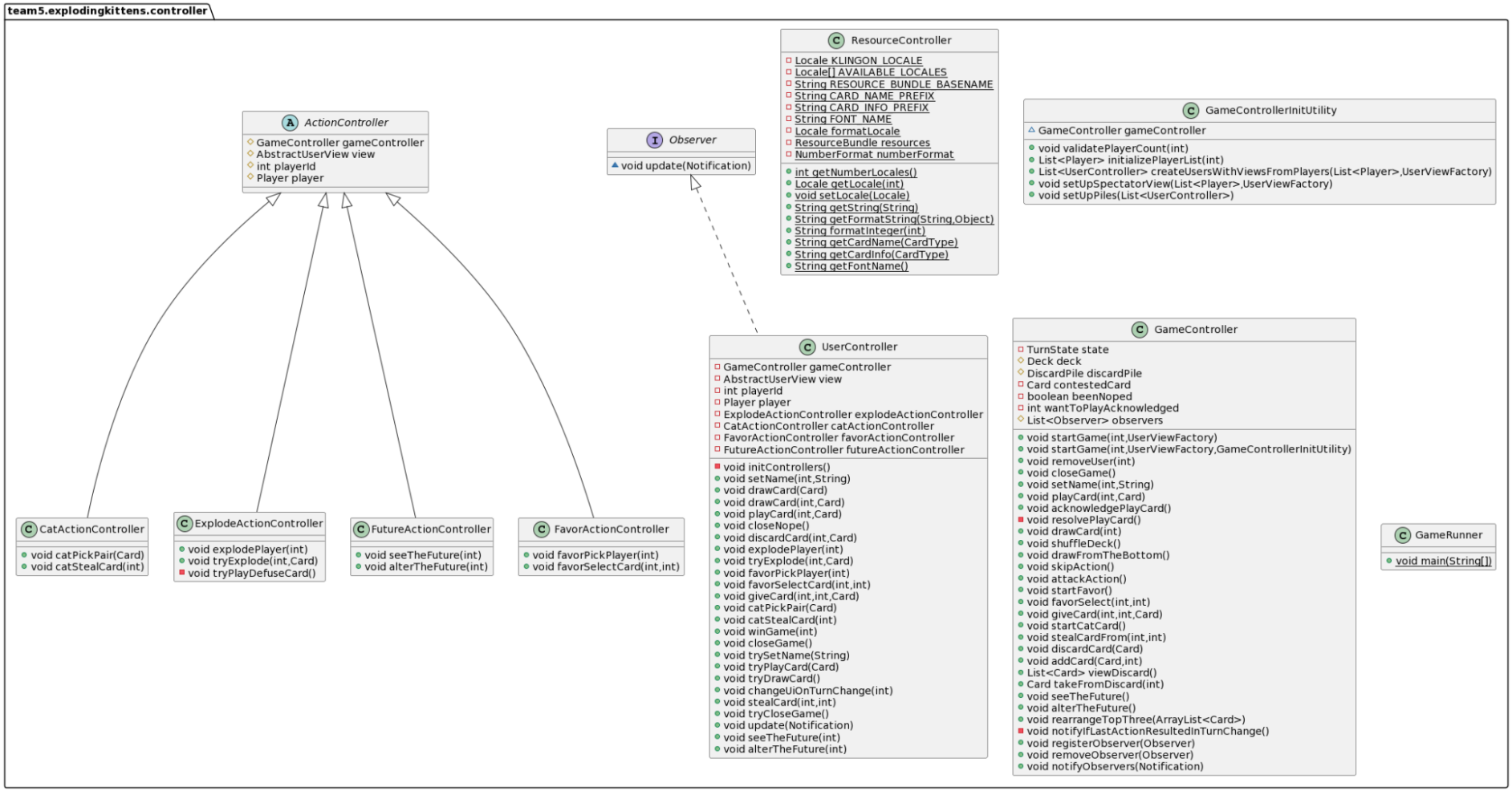
- F01: Ability to view starting screen
- F02: Tracks multiple screens (1 for each player)
- F03: Ability to play with up to 10 players
- F04: Ability to play in English
- F05: Ability to play in Japanese
- F06: Ability to play in Klingon
- F07: Tracks the deck
- F08: Tracks the discard pile
- F09: Tracks a players' hands
- F10: Tracks a player's selected cards
- F11: Ability to draw cards
- F12-F20: Ability to play the following cards with varying effects:
    - F12: Attack: End turn, force next player to take two turns
    - F13: Skip: End turn immediately
    - F14: Cat: Play as a pair to steal a card from a player of their choosing
    - F15: Defuse: Avoid losing the game by defusing an Exploding Kitten
    - F16: Nope: Cancel another player's special card
    - F17: See the Future: Peek at the next three cards in the draw pile
    - F18: Shuffle: Shuffle the deck
    - F19: Favor: Choose a random card from an enemy's hand
    - F20: Draw from Bottom: Draw a card from the bottom of the deck
- F21: Ability to eliminate a player by drawing an Exploding Kitten card
- F22: Ability to determine a winning player to be the last one standing

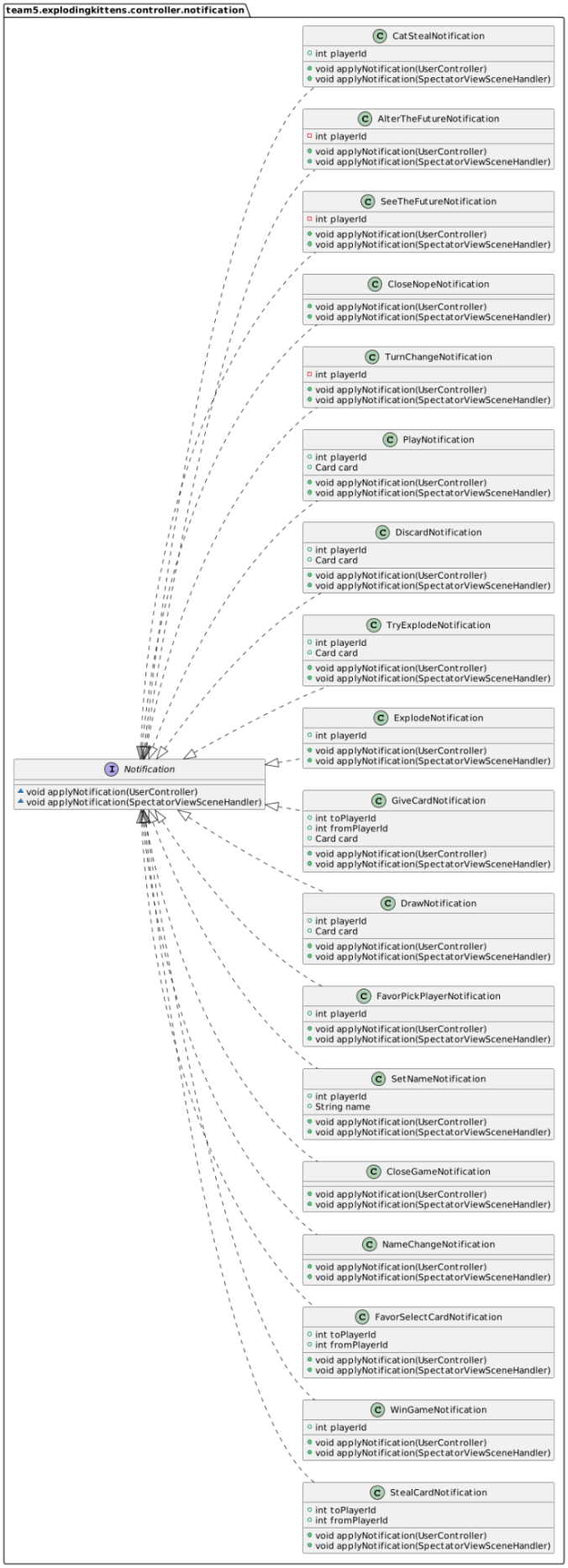
Software Architecture and Design

A full software architecture and design UML diagram can be accessed here.

This design opted for a model-view-controller oriented design, with each being separated into its own package. For ease of viewing, the design UML Diagram has been broken up into a package-level view below. Inter-package dependencies are not shown.The View package was also omitted from the following images due to size and formatting difficulties.




team5.explodingkittens.controller
ActionController
GameController gameController
AbstractUserView view
int playerId
Player player
Observer
void update(Notification)
ResourceController
Locale KLINGON_LOCALE
Locale[] AVAILABLE_LOCALES
String RESOURCE_BUNDLE_BASENAME
String CARD_NAME_PREFIX
String CARD_INFO_PREFIX
String FONT_NAME
Locale formatLocale
ResourceBundle resources
NumberFormat numberFormat
int getNumberLocales()
Locale getLocale(int)
void setLocale(Locale)
String getString(String)
String getFormatString(String,Object)
String formatInteger(int)
String getCardName(CardType)
String getCardInfo(CardType)
String getFontName()
GameControllerInitUtility
GameController gameController
void validatePlayerCount(int)
List<Player> initializePlayerList(int)
List<UserController> createUsersWithViewsFromPlayers(List<Player>,UserViewFactory)
void setUpSpectatorView(List<Player>,UserViewFactory)
void setUpPiles(List<UserController>)
CatActionController
void catPickPair(Card)
void catStealCard(int)
ExplodeActionController
void explodePlayer(int)
void tryExplode(int,Card)
void tryPlayDefuseCard()
FutureActionController
void seeTheFuture(int)
void alterTheFuture(int)
FavorActionController
void favorPickPlayer(int)
void favorSelectCard(int,int)
UserController
GameController gameController
AbstractUserView view
int playerId
Player player
ExplodeActionController explodeActionController
CatActionController catActionController
FavorActionController favorActionController
FutureActionController futureActionController
void initControllers()
void setName(int,String)
void drawCard(Card)
void drawCard(int,Card)
void playCard(int,Card)
void closeNope()
void discardCard(int,Card)
void explodePlayer(int)
void tryExplode(int,Card)
void favorPickPlayer(int)
void favorSelectCard(int,int)
void giveCard(int,int,Card)
void catPickPair(Card)
void catStealCard(int)
void winGame(int)
void closeGame()
void trySetName(String)
void tryPlayCard(Card)
void tryDrawCard()
void changeUiOnTurnChange(int)
void stealCard(int,int)
void tryCloseGame()
void update(Notification)
void seeTheFuture(int)
void alterTheFuture(int)
GameController
TurnState state
Deck deck
DiscardPile discardPile
Card contestedCard
boolean beenNoped
int wantToPlayAcknowledged
List<Observer> observers
void startGame(int,UserViewFactory)
void startGame(int,UserViewFactory,GameControllerInitUtility)
void removeUser(int)
void closeGame()
void setName(int,String)
void playCard(int,Card)
void acknowledgePlayCard()
void resolvePlayCard()
void drawCard(int)
void shuffleDeck()
void drawFromTheBottom()
void skipAction()
void attackAction()
void startFavor()
void favorSelect(int,int)
void giveCard(int,int,Card)
void startCatCard()
void stealCardFrom(int,int)
void discardCard(Card)
void addCard(Card,int)
List<Card> viewDiscard()
Card takeFromDiscard(int)
void seeTheFuture()
void alterTheFuture()
void rearrangeTopThree(ArrayList<Card>)
void notifyIfLastActionResultedInTurnChange()
void registerObserver(Observer)
void removeObserver(Observer)
void notifyObservers(Notification)
GameRunner
void main(String[])


team5.explodingkittens.controller.notification
CatStealNotification
int playerId
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
AlterTheFutureNotification
int playerId
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
SeeTheFutureNotification
int playerId
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
CloseNopeNotification
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
TurnChangeNotification
int playerId
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
PlayNotification
int playerId
Card card
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
DiscardNotification
int playerId
Card card
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
TryExplodeNotification
int playerId
Card card
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
ExplodeNotification
int playerId
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
Notification
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
GiveCardNotification
int toPlayerId
int fromPlayerId
Card card
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
DrawNotification
int playerId
Card card
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
FavorPickPlayerNotification
int playerId
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
SetNameNotification
int playerId
String name
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
CloseGameNotification
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
NameChangeNotification
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
FavorSelectCardNotification
int toPlayerId
int fromPlayerId
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
WinGameNotification
int playerId
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)
StealCardNotification
int toPlayerId
int fromPlayerId
void applyNotification(UserController)
void applyNotification(SpectatorViewSceneHandler)

Testing Strategy

The testing strategy for this product primarily relied on utilizing regression tests to make sure that existing behavior was preserved with changes, and a combination of automatic and manual testing approaches were applied to the newly implemented features. Manual testing was used where behavior of the JavaFX based UI needed to be observed, and automatic tests for UI could not be run by continuous integration.

This product was developed using a strict TDD discipline, combined with code coverage tools and style checkers. A definition of correctness and done was determined through boundary value analysis, and those values were asserted in unit tests using the JUnit library. Code coverage and mutation testing was assessed using Pitest. A detailed description of every quality assurance process is given below.

Test Driven Development Cycle

A test driven development strategy was taken in order to verify correctness of model-level classes and behavior. BVA was the primary tool for determining the definition of done, and was followed for all of the objects that were written in the Model and Controller packages. Because of difficulties of having Continuous Integration run UI testing libraries, the manual testing approach was used to verify behaviors in the View package.

Unit Testing

Unit testing was conducted on the same two packages (Model, Controller) as TDD, and was conducted with the aid of the JUnit and EasyMock libraries. JUnit was used to verify assertions and exceptions, and EasyMock was used to verify the integrity of unit tests by mocking and classes external to the scope of a given unit test. A large majority of the testing suite is composed of these unit tests, which were designed and finalized before this project was iterated upon, and these tests are regression tests for our purposes.

Integration/System Testing

Integration and systems tests were primarily conducted manually, as the primary classes under these tests were involved with the JavaFX, and could not run correctly under Continuous Integration. The tests for each feature in each iteration was primarily tested in a scripted, manual approach where a predetermined set of actions was taken and input into the UI, and the resulting behavior was manually verified by the manual tester. A few UI tests exist in the View package of the test suite, but while these run correctly on a local machine, the CI runner for our version control software (GitHub) does not run these tests correctly due to its machine configurations, so these tests may not be present in our final delivery.

Code Coverage

Code coverage was assessed on the basis of line coverage. Code coverage was performed by the Pitest tool, and was performed over the Model and Controller package. Code coverage for the Model and Controller packages was assessed to be at 91% and 94% respectively. The remaining percentages in these packages indicate dependencies and calls to View package classes which could not be put under a unit test and run with CI.

Mutation Testing

Mutation testing was assessed also with the aid of the Pitest tool running the “default” mutator operation group. The “default” mutator group includes the creation and verification of the following mutant types:

- Conditionals Boundary
- Increments
- Invert Negatives
- Math
- Negate Conditionals
- Void Method Calls
- Empty returns
- False Returns
- True returns
- Null returns
- Primitive returns

Mutation coverage was again assessed over the Model and Controller classes giving 94% and 90% coverage respectively. Again, the remaining coverage indicates that View package classes dependencies were intentionally not covered by mutation testing, as these would be tested manually as described above.

Testing Suite Summary

The testing suite summary is broken up into unit and system/integration test descriptions below. Unit testing was conducted entirely in a scripted, automated manner, and system/integration tests were written in a combination of manual and automatic scripted testing.

Unit Test Descriptions

The unit test suite is broken up into packages-level tables below, and into regression and manually tested sections:

Regression Tests

Package: Controller			
Test Class	Class(es) Under Test	Methods Tested	Pass / Fail

NotificationTests	AlterTheFutureNotification CatStealNotification CloseGameNotification CloseNopeNotification DiscardNotification DrawNotification ExplodeNotification FavorPickPlayerNotification FavorSelectCardNotification GiveCardNotification NameChangeNotification PlayNotification SeeTheFutureNotification SetNameNotification StealCardNotification TryExplodeNotification TurnChangeNotification WinGameNotification	applyNotification()	Pass
GameControllerTests	GameController	startGame() addCard() closeGame() removeUser() drawFromTheBottom() discardCard() viewDiscard() takeFromDiscard() shuffleDeck() skipAction() attackAction() setName() startFavor() favorSelect() giveCard() startCatCard() stealCardFrom() acknowledgePlayCard() seeTheFuture() alterTheFuture() rearraneTopThree()	Pass
SubjectTests	Subject	registerObserver() removeObserver() notifyObservers()	Pass
UserControllerTests	UserController	drawCard() playCard() tryPlayCard() update() trySetName()	Pass

		tryDrawCard() tryCloseGame() catPickPai() catStealCard() catStealCard() stealCard() closeNope() discardCard() seeTheFuture() alterTheFuture()	

Package: Model			
Test Class	Class(es) Under Test	Methods Tested	Pass / Fail
CardActionTests	CardAction	applyAction()	Pass
CardTests	Card	getName() checkForExplodingKitten() checkForDefuse() checkForCatCard() isSameCatTypeAs() getImagePath() getCardInfo() playCard()	Pass
CardTypeTests	CardType	getName() getBasePath() getPath() getCount()	Pass
DeckTests	Deck	getSize() shuffle() draw() drawAtBottom() insertCard()	Pass
DiscardPileTests	DiscardPile	discardCard() takeCard() viewCards() size()	Pass
PlayerTests	Player	getCard() addCard() size() removeCard() removeAllCards() getCardType()	Pass

		hasCardType() hasMatchingPair() findMatchingTypes() findCardOfTypeExcluding() getRandomCard()	
TurnStateTests	TurnState	drawCard() attackAction() skipAction() incrementTurn() invalidatePlayer() explodeAction() findLowestValid()	Pass

Note: The View package tests must be removed from the “exclude” clause in the build.gradle file and run locally, as GitHub’s continuous integration runner is incompatible with the testing library used for this package.

Package: View			
Test Class	Class(es) Under Test	Methods Tested	Pass / Fail
DialogTests	LanguageFriendlyChoiceDialog LanguageFriendlyEmptyDialog LanguageFriendltInputDialog ChangeImageDialog	getDialogPane()	Pass
FututeAlteringDialogTests	FutureAlteringDialog	chooseNewOrder()	Pass
SpectatorViewTests	SpectatorView	getScene()	Pass
UserViewSceneBuilderTests	UserViewSceneBuilder	populatePlayerUiList()	Pass
UserViewTests	UserView	getScene() changeUiOnTurnChange()	Pass

Manual Tests

Scripted manual testing was conducted over features rather than classes, in order to test new functionality that was only visually testable through integration testing.

Features			
Feature	Relevant Class(es) Under Test	Test Description	Pass / Fail
Card Sorting	UIPlayerHand	The aim of this feature was to sort cards alphabetically in the hand, so cards were tested and manually assessed to be alphabetized at the following points in gameplay: - The first turn - After drawing a card - After playing a card - After selecting a card - After hovering on a card The image corresponding to the card and its effect when played was also verified.	Pass
Card Function Display	UiDeck, UiDiscard	Add labels to the various UI components to better indicate their purpose and differentiate themselves. For example, although the discard pile, deck, and elements for other players all have the same image, they now have different labels to help set them apart.	Pass
Turn State Display	UserView, UserViewSceneBuilder, UserViewSceneHandler	This feature displays to each user information regarding their turn state. Specifically, the window informs the user that they are still waiting for their turn (if it is not currently their turn) or that they are able to perform an action now (if it is currently their turn). The feature was tested for 2-10 players with enumerated names (e.g. “Player1”), with each player’s turn consisting of one of the following actions: - Draw a card	Pass

		- Play a Draw From The Bottom card - Play a Skip card - Play an Attack card These actions cause the game to transition to the next player's turn. An entire cycle of turns was performed to verify that only the player with the current turn had a display notifying them of their status.	
Custom Card Art	ChangeImageDialog, StartView	Verify that an image can be uploaded to the system and the type of card that they'd like their image to be associated with can be specified. On the main screen of the game, before pressing start, the user can select "Change Card Images" to customize their cards. A dialog asks for which card to change the image of. It will prompt for a new .png file to upload. From then on, whenever the game is played, it will use the most recently uploaded image for that given card.	Pass
Pass and Play	UserView	The dominant view on the screen automatically changes to the view associated with the player whose current turn it is. This is already true during opportunities where a player may perform a counteraction (e.g. player is able to "Nope" an action), as new dialogs automatically are brought to the front. The current window is brought to the front so as to keep other windows visible, should players wish to view	Pass

		them separately or multiple at a time.	
Alter the Future	CardType, FutureAlteringDialog, Card	In our starting code, the “alter the future” card was not fully implemented. Much of the code was already there, but nothing happened when it was played; everything was linked up and the card is not implemented as specified in the rules.	Pass
Key Bindings	UserViewSceneHandler UserView	The feature implemented gave functionality to pressed keys, and the following key-action pairs were tested: - Numeric keys to select a card 1 through 10 in the hand - Pressing a numeric key twice to select and further deselect a card - Pressing SPACE or ENTER to play a selected card - Pressing SHIFT in a valid turn to draw a card - If ENTER or SHIFT was pressed without a card selected, no cards would be played - If SHIFT was pressed while not on a player’s turn, a card would not be drawn	Pass
Spectator Mode	SpectatorView, SpectatorViewSceneBuilder, SpectatorViewSceneHandler, SpectatorViewSinglePlayerUI	The system generates a view of the game that displays more information than available to the participants. The view gives equal space for up to 10 players to showcase their player name and the cards in their hand, which is updated after each turn transition. This feature was tested for 2-10	Pass

		players, confirming that the grid layout system allocates space for every count of players appropriately. The following actions were performed to test if updates were sent to the spectator view: - Name a player - Draw a card - Play a card The manual test passes if the spectator view successfully updates the displayed information based on the change in state caused by all three actions.	