http://casino308.com/

Meta Tags

Home

 Welcome to twenty-one's final project for CS308! This website serves as a user manual to navigate how to *interact* with the game and *extend* it. The buttons on the top of the screen link to information regarding various components of the gameplay. There is first the overview, which describes the capabilities and flow for user interaction. There are also links to the Lobby, Validator, and Gameplay sections, which are more specific and tied to certain functionality.

Additionally, on the left side of the screen, you will see links to specific XML file types. One file, a LobbyView, is needed to launch a game. From there, a bundle of five files—including a Deck, Game, Hand, Players, and GameView—runs a specific card game. By explaining how to interact with our game, in addition to its underlying data, our hope is that future developers can create custom games that extend its flexibility without changing the underlying architecture.

Overview

 To launch our program, run the *Main* class, which can be found in the *ooga* package. In this class, there is a constant String which provides the path to the first XML file. This is a *LobbyView* file. This XML file type lists a variety of information for the main screen of the game, including "file bundles" that encode preloaded games (such as blackjack or poker). To change the initial lobby, change this instance variable.

Upon runtime, the Lobby will be loaded, which will show the various games to play. A simple mouse click on any of the loaded games will then begin the initialization process for the selected type. Immediately, a new stage should appear on the left, which will list the status of the five icons selected. If all five are valid (indicated by the valid image file given via a resource bundle), then the launch button will be enabled. On click, the game will be rendered.

If the files are *not* valid, then this button should be disabled. The files that are either empty or invalid will be marked as such, and in doing so, a new *Multiple XML Chooser* will appear to its right. This will allow the user to continue to select new files until all five are valid. The display will update real time as this process goes on. After they are all marked as valid, the initialization button will be enabled. Upon click, the last phase of the UI will emerge in a new, centered screen.

This GameView will encompass the functionality of a real-life casino game. It will break the gameplay up into individual actions, specified by the user who is the *Main Player* (located on the bottom of the screen). Whenever an action is prompted, assume it is for this player, whose name and bankroll are attached to the displayed cards. As the game goes on, bankrolls will be updated and cards will be dealt and cleared according to the logic of the XML file.

The top of the GameView and LobbyView also display a Settings Bar, which allows in-game customization from a variety of options. The language can also be changed as well. Lastly, a help Icon can be found, which links to this site, explaining the possible user interactions.

Lobby

 The *Lobby* presents the user with a *View* of the games that he can play, as dictated by the *LOBBY_XML_FILE* instance variable in the *Main* class. This file encodes the necessary information to construct a home page for our application. It specifies the possible game variations to offer, the languages to offer them in, and the stylization of the lobby itself. It further specifies the width and height of this stage, which is centered in the user's display.

In the Lobby, one can select a game by clicking on the *Game Starter* that contains the name and icon of each possible game. Doing so will launch the Validation process, which is defined under the Validator section of this page. This click represents the "loading" of XML files into the game engine, which can then create a *Game View* once they pass *Validation*.

Additionally, there is a *Settings Bar* on top of the *Lobby View*. This allows the user to change the language of the game and the style. The Info button also links back to this site.

Validator

The phase between game selection and game initialization is *Validation*. This is a *mandatory* process, and games *cannot* be loaded without having every file checked. This is crucial to prevent crashes spurred by errors and exceptions.

As such, when a game type is selected in the lobby, a new *All Files Display* is created which validates each file individually. At initialization, the files preloaded from the Game Type are checked and their statuses are displayed. The icons for the file type, the equality sign,

and the status displayed are all data-driven from ResourceBundles. In our project, these can be found in the fileStatuses and fileTypes packages which are in iconBundles.

If any files are deemed invalid, the user will be presented with a *Multiple XML File chooser. This will allow the user to select any number of files and then submit them. The validator will accept the first valid file using a descending order, and if none are valid, then it will mark the files as invalid. Any given file exists in one of three enumerated states: *Valid, Empty, Invalid*.

Once all five files are marked as valid, a launch button will become enabled. Upon click, the desired Game View will be rendered. If for any reason, the user wants to leave the validation process and select a new game, the *All Files Display* can simply be exited out of.

Gameplay

 The third stage of playing a card-based game is, well, actually playing the game! This occurs in a *Game View* object, which is rendered from five different XML files. A specific file is needed for the deck; the winning and losing hands; the game logic; the number of players; and the view itself. All these come from the Lobby and Validation processes. The Game View then translates the data to a GUI that can communicate with the backend and handle user input.

 In the Gameplay, there are a variety of components to be considered. The first are the *players*. These represent the users in the game, and are constructed from a single XML file. Each player has a name, bankroll, and collection of Bets, which individually store cards and a wager. One player can act at a time, and the Game shows the current player whose turn it is by placing them on the bottom of the screen in a *Main Player View*. The *Other Players* are displayed on the left side and rotated into the spotlight automatically.

 In many poker games, there are often cards that do not belong to any individual player. These can be common cards or dealer cards, and are rendered in the center of the screen. The dealer is named an *Adversary* in our game and is represented only with a *Hand View*. The common cards are likewise represented with just a *Hand View*. If the game is poker, a *Pot View* will further be rendered on the right side of the screen. This will show the current amount of money in escrow that will be afforded to the winner of the hand.

 At the end of a turn, the GUI will notify the winners and losers by setting their colors to winning and losing hands. These colors come from a CSS file in data, so it *is* possible that no change will occur. If this is the case, simply change the CSS file. In general, if there is ever a

perceived bug in the UI coloring or formatting, a good method is to change the stylesheet via the chooser in the Settings Bar. These styles affect the spacing and design of the layout.

 When a player is prompted for action, this occurs via Dialogues that are presented and intended to be responded to by the Main Player. There are two dialogues: Wager Selectors and Action Selectors. They simply prompt the user for input regarding an amount to place on a hand, or they prompt the user to select an action on their hand (such as ask for a new card). A player should hit the *Ready* button that is presented to signify that they are ready to give input. Note that there is error handling embedded in the Dialogue, so entering a non-numeric wager, for example, will be prevented.

 If an exception occurs due to input, a screen will pop up informing the user of the error. These exceptions are likewise stylized from the XML GameView file, where a style sheet is referenced for Errors.

 The Game will run until a Goal is met. As of now, there are tournaments and cash games. Cash games will run forever, and tournaments will run until there is only one winner. If the user wishes to exit before reaching a goal, he can simply exit out of the Game View screen. After doing so, the Lobby View will still be rendered, so a new game can be launched.

XML Files

Deck

 The deck XML file contains a sequence of *Card* tags which encompass a *Suit* and a *Value*. The *Value* is always a parseable double and the *Suit* can be any string. Additionally, there are a few meta tags which provide more information about the file and deck makeup.

Each deck has an (optional) *Title* and *Author*. Additionally, there is a *Quantity* tag which specifies the quantity of the deck in the dealer's possession (e.g. in many casinos, multiple decks will be used to avoid constant shuffling at larger tables).

There is also a *Type* tag, which specifies the type of deck. The two options are Remove and Replace. A Remove deck has cards removed as they are dealt out, whereas a Replace deck has cards replaced after each round. More traditional card games are of the Remove format, however to accommodate games such as Roulette and Craps where a "Card" object never truly disappears from gameplay, they must be replaced back in the Deck once selected.

Game

 The Game XML file contains information for the specific game and facilitates creation and parametrization of the Controller object. The first section encodes information regarding the game *Type*. Right now, we support an *Adversary* game (where all players go individually against a dealer) and *Group* games (where all players are competing against each other). This tag facilitates reflection within the ControllerFactory to create the appropriate type of Controller which runs the game logic. Additionally, Competition Types have a required parameter(s). *Adversary* games require a minimal hand which the dealer will play through, and *Group* games require an ante. The next two tags encode *TableMin* and *TableMax*, which are both parseable doubles and bracket bet sizing.

Next, there is a *CardShow* status, which dictates how cards are displayed in the view. Currently, there are three types of CardShow, All (all are shown), Active (only acting player is shown), and Other (all cards except active player are shown, aka you have no idea what your cards are). Then, the final meta tag is a *Goal* tag, which drives the round to round structure within the game. Currently, support is offered for Tournaments (play until one player left) and CashGames (play indefinitely, goal is the most money). Both *CardShow* and *Goal* tags are validated by enumeration in the Game XML schema, and have delegated responsibility with lambdas from the controller.

The final bundle drives the sequence of events within the game. All casino games start with an *EntryBet* (validated by enumeration, currently Generic and Specific exist). After an initial bet has been placed (you gotta pay to play), there is an alternating series of Dealer and Player actions.

Hand

 The Hands XML file has two sections - Winning and Losing hands. These hand types have a bundle of tags which include a *Name* to facilitate reflection, *Parameters* to maintain the general nature of hands (e.g SumUnderX name with parameter 22), *Multiplier* for winning hands which modulates payouts, and an optional *ViewName* so the frontend doesn't see the more code friendly *Name* tag.

Furthermore, the validation schema ensures that the *Multiplier* is a parseable double, and the *Name* is one that we have created a class for to mitigate the risk of ReflectionExceptions at runtime. The *Parameter* tag is composed of space separated doubles. Schema validation was fairly difficult, so protection for flawed *Parameters* happens in the backend. Defaults are given for any *Parameters* that cannot be parsed as doubles or that are expected but do not exist.

Players

 The Players XML file is fairly straightforward. It currently has three tag types. *Player* separates individual players. Each *Player* tag requires a *Name* tag (which is a parseable string), and a *Bankroll* tag which is a parseable double.

Gameview

 The GameView XML file drives the construction of the specific game view. It has two meta tags which encode the game *Title* and file *Authors*. Additionally, it encodes the starting dimensions of the screen in the *Width* and *Height* tags.

The remaining tags encode information to compose the SettingsBar as well as visual effects internal to the Game. There are *Language* and *Stylesheet* tags which fill the ComboBox dropdowns on the top of the screen. Additionally, there is an *IconBundle* and *ErrorStyleSheet* which encode visual components of the game. All of these tags which point to specific data files/bundles are validated by enumerated types internal to the view schema.

Lobbyview

The LobbyView XML file contains meta tags for starting *Dimensions (**Width* and *Height*). Similar to the GameView XML file, it also contains *Stylesheets* and *Languages* which drive the creation of the SettingsBar on top. There is also an *IconProperties* bundle which dictates the icons in the Settings bar, as well as an *ErrorStylehseet* reference.

The next section contains two tags which drive the IconBundles contained within the Validation view (*FilesStatusBundle* and *FilesIconBundle*). These customize the appearance of the validation view and are validated with enumerated types internal to the schema.

The final section contains a series of *Bundle* tags which encode information that creates Icons seen in the LobbyView. Each Icon contains a *Name*, an *Icon*, and a sequence of *Files (Deck, Game, Hands, Players, and View)*. All of the icons are validated by enumerated types in the schema, and the file names are validated in the validation view sequence. Each bundle must include a *Name* and an *Icon* tag, and then contains an arbitrary number of files. The rest of file selection happens in the Validation loop.
