Project 7
Terminal Casino

Dananjay Srinivas, Weitung Liao, Jay Seabrum

Link to Repo:

On the off chance that the link to the repo is lost, here it is:
https://github.com/fortytw0/LING5448-OOAD/tree/main/Project6

Note that the link to the Project 7 Repo is the same link to the Project 6 Repo. We did this to make it easier for us to update/push code leading up to the submission of this sprint.

Link to Video Demonstration:

On the off chance that the link to the video demonstration is lost, here it is:
https://www.youtube.com/watch?v=1i2t0i_H9W8

There are timestamps in the YouTube comments and description of the video for ease of navigation and access. If they do not work for any reason, then they are replicated here below:
00:00 - Introductions and Requirements for Executing the Code

00:37 - Terminal Casino Demo
Login/Main Menu (00:50)
Blackjack (01:05)
Settings (02:11)
Slots (02:39)
Roulette (03:06)
Logout (03:38)

03:40 - Responsibilities of the Members in the Project
Dananjay (03:49), Jay (09:27), Weitung (12:28)

16:57 - Reflection
Dananjay (17:08), Jay (19:59), Weitung (20:35)

Final State of the System:

From the original goals of the project, the following parts were implemented:

- Implementing the MVC pattern (custom built)
- The main casino games: BlackJack, Roulette, Slots
- Implementing the main controllers (ie: main menu, settings, log in, log out, and the three games above) and successfully switching between them using a combination of the State and Command patterns.
- Implementing a Mongo database as an ORM to keep user login and wallet information persistent (this is the Model branch of the code) when switching between controllers.

The following parts were not implemented:

- The Decorator pattern was never implemented within Roulette as it wasn't needed in the end. It was found to be easier to have a list and loop over that list instead.
- A feature to keep track of the user's Win/Loss ratio per game (and to display this in the Settings screen) was not implemented simply because it wasn't a major feature that needed to be implemented for the completion of the project.
- Password masking when logging in. Unfortunately, when the user types in their password, it is visible in the console. This wasn't implemented as it was relatively minor compared to other parts of the project.

Third Party Code vs Original Code Statement:

The main third party code that we utilized was the MongoDB API for the user model. Additionally, the ABC library documentation was referenced much and used as a rough guide so as to make sure that we could code and create our own MVCs from scratch correctly. Besides that, all other code within this project is original to the authors listed above.

General overall OOAD process doing the Semester Project:

- Designing the MVC framework from scratch was an early roadblock. While this was an initial roadblock, figuring it out for ourselves and getting it to work with our project was a great learning experience and great to do overall.
- Views and aesthetics of how the user interacts with our code (ie, even though the game is in the terminal, at least designing it to look good to the user is a UI approach to designing the code).
- Major priorities took over (minor stuff didn't happen such as password masking, other minor things)
- Inspired more by a BDD approach than a TDD approach while coding.
- While not directly implemented (ie, Unit Tests), the process of designing the code with a test focus in mind kept the code more cohesive and less coupled overall.
- The abstract controllers, models, and views we created helped us to code with a contract in mind. Additionally, because we decided to use MongoDB and can't change MongoDB's internal code and structure, we were forced to code with this contract in mind.
- We tried to minimize code smells as much as we could throughout the project (ie. YAGNI, DRY)
- In the BlackJack controller, the code is cohesive, but due to the calls to the View, it feels as though there is a lot of coupling between the controller and the view, but this is so that the print statement happens so as to give the user the appropriate feedback.

UML Diagrams:

For the finalized project, these were the patterns that were used:

MVC - This pattern served as the basis for the entire project. The Model was used to handle the user's information and to persist that information until the user logged out. The Views were

primarily used to print to the console when needed. The Controllers were used to contain the logic for each of the games and screens.

Factory - This pattern (used in conjunction with Command) was primarily used to return the correct controller when needed.

State - This pattern was used to shift seamlessly between the multiple controllers within the project.

Prototype - Under BlackJack, there are prototype objects under the Constants classes using the in-built Python command .copy(), we can clone them for their various uses in the game.

Strategy - All code that wasn't directly part of the main games (BlackJack, Slots, Roulette) or part of the main controllers (Main Menu, Log in, Log Out, Settings) were called when needed. Examples of util code in the project would be num_print, which formats numbers to 2 decimal places, or CheckFloat which checks to see if the input is a float or not. Calling these util functions is a classic case of using Delegation/Strategy to keep the code as OO as possible.

Command - This pattern was used to switch between the controllers. It is also used when asking the user if they want to play the game that they had selected again, or if they wanted to go back to the main menu.

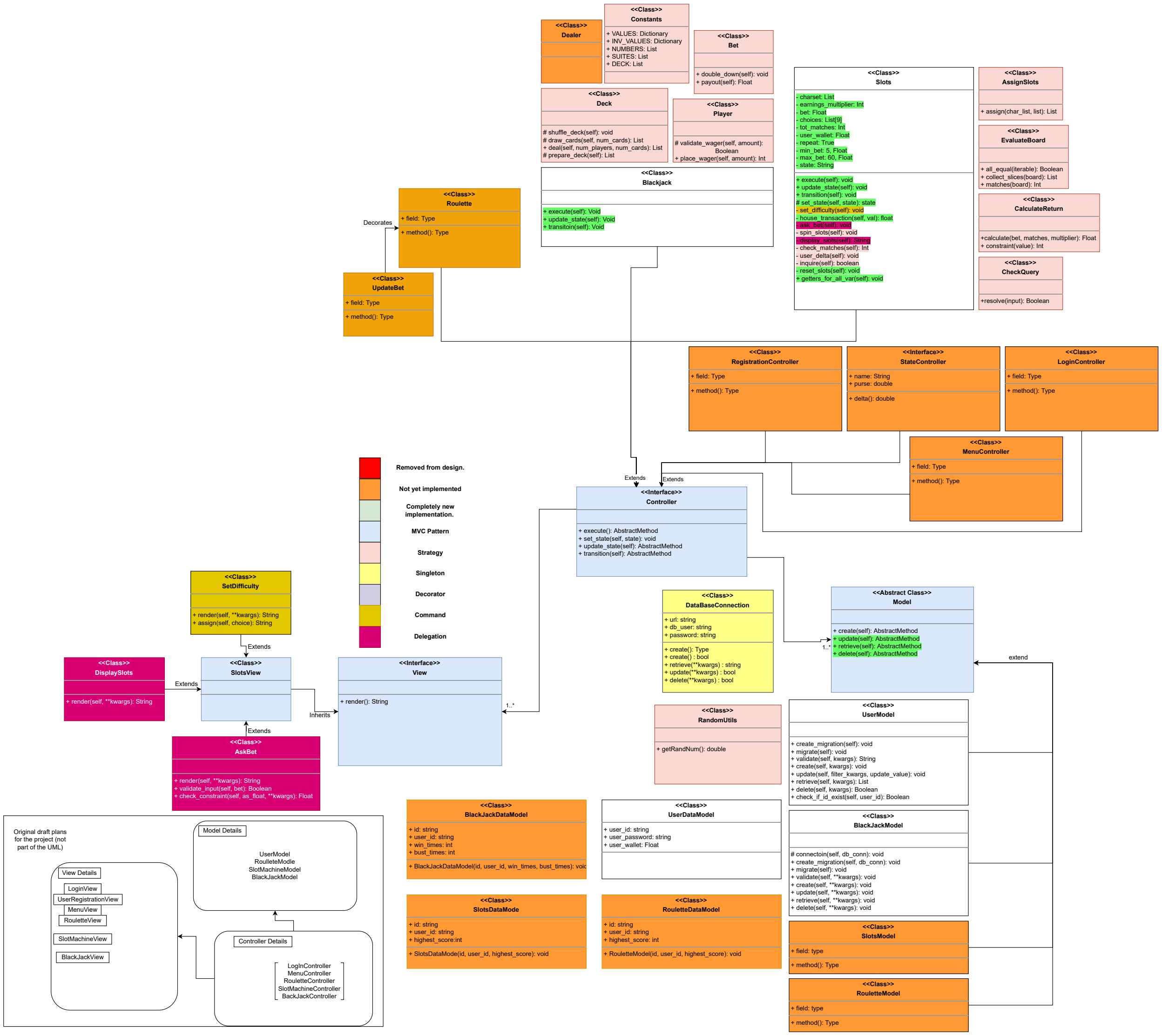
These are the key changes that happened between Project 5/6 and Project 7 with regards to the UML diagram:

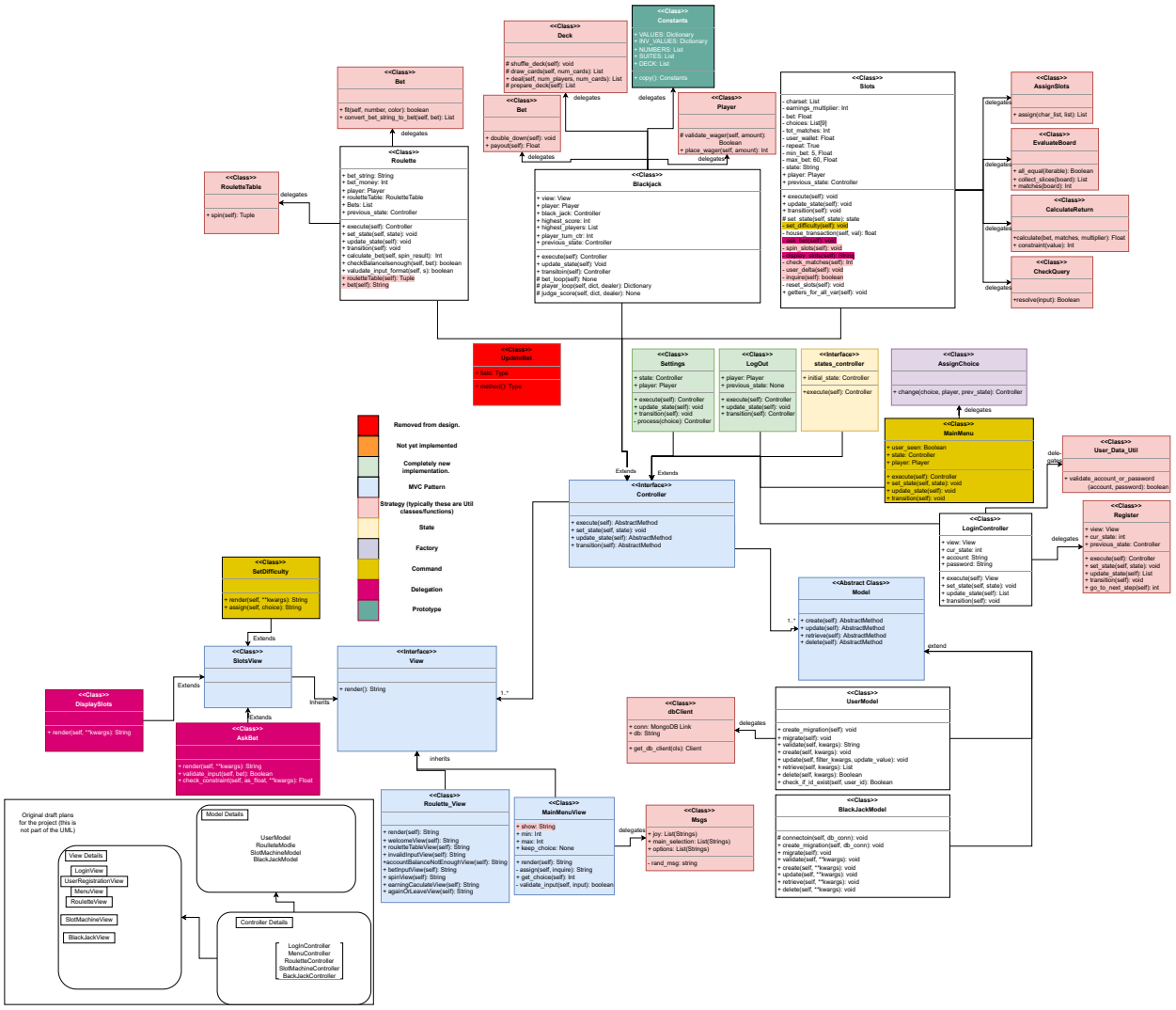
- The following classes in the Project 5/6 UML diagram have been removed/changed:
    - SlotsModel (removed)
    - RouletteModel (removed)
    - RouletteDataModel (removed)
    - BlackJackDataModel (changed to the 4 different classes that the BlackJack Controller delegates/strategizes to)
    - RandomUtils (changed; different controllers implement their own utils where needed. These are demarcated as delegations/strategy in the Project 7 UML)
    - DataBaseConnection (changed; this was originally going to be the Singleton pattern in the project, but is now delegated to. It has been reformed into the dbClient class in the Project 7 UML)
    - UpdateBet (removed; this Decorator pattern was supposed to be part of Roulette. Was scrapped for a simpler Bet class and that class was then delegated to)

- The following classes are present in the Project 7 UML diagram but were not in the Project 5/6 UML:
    - Roulette_View and MainMenuView are added and inherit directly from the abstract View class.
    - Msgs is a helper class that is delegated to from MainMenuView

- Settings is a brand new class that extends from the abstract Controller interface. By its namesake, it allows the user to check their settings
- LogOut is a new class that… logs the user out and exits the program. It extends the abstract Controller interface.
- States_Controller is the main mover of the project. It is the State pattern in our project. It helps transition between the different states (controllers) in the project.
- The MainMenu controller has been changed to be the Command pattern
- AssignChoice is a new class that is delegated to from the MainMenu. It is the Factory pattern in our project.
- The LogIn Controller has two new classes that it delegates to: User_Data_Util and Register.
- The details for the Roulette Class have been filled out when compared to the Project 5/6 UML diagram. It now delegates to two other classes, Bet and RouletteTable
- The details for the BlackJack class have been altered. It now explicitly shows the strategy/delegation that the class uses. Furthermore, the CONSTANTS class that the BlackJack class delegates to is the instance of Prototype within the project.

And finally, the following two pages of this document are the UML diagrams for Project 5/6 then Project 7.


<<Class>>
Dealer
<<Class>>
Constants
+ VALUES: Dictionary
+ INV_VALUES: Dictionary
+ NUMBERS: List
+ SUITES: List
+ DECK: List
<<Class>>
Bet
+ double_down(self): void
+ payout(self): Float
<<Class>>
Deck
shuffle_deck(self): void
draw_cards(self, num_cards): List
+ deal(self, num_players, num_cards): List
# prepare_deck(self): List
<<Class>>
Player
validate_wager(self, amount): Boolean
+ place_wager(self, amount): Int
<<Class>>
Blackjack
+ execute(self): Void
+ update_state(self): Void
+ transitoin(self): Void
<<Class>>
Roulette
+ field: Type
+ method(): Type
Decorates
<<Class>>
UpdateBet
+ field: Type
+ method(): Type
<<Class>>
Slots
- charset: List
- earnings_multiplier: Int
- bet: Float
- choices: List[9]
- tot_matches: Int
- user_wallet: Float
- repeat: True
- min_bet: 5, Float
- max_bet: 60, Float
- state: String
+ execute(self): void
+ update_state(self): void
+ transition(self): void
# set_state(self, state): state
- set_difficulty(self): void
- house_transaction(self, val): float
- ask_bet(self): void
- spin_slots(self): void
- display_slots(self): String
- check_matches(self): Int
- user_delta(self): void
- inquire(self): boolean
- reset_slots(self): void
+ getters_for_all_var(self): void
<<Class>>
AssignSlots
+ assign(char_list, list): List
<<Class>>
EvaluateBoard
+ all_equal(iterable): Boolean
+ collect_slices(board): List
+ matches(board): Int
<<Class>>
CalculateReturn
+calculate(bet, matches, multiplier): Float
+ constraint(value): Int
<<Class>>
CheckQuery
+resolve(input): Boolean
<<Class>>
RegistrationController
+ field: Type
+ method(): Type
<<Interface>>
StateController
+ name: String
+ purse: double
+ delta(): double
<<Class>>
LoginController
+ field: Type
+ method(): Type
<<Class>>
MenuController
+ field: Type
+ method(): Type
Removed from design.
Not yet implemented
Completely new implementation.
MVC Pattern
Strategy
Singleton
Decorator
Command
Delegation
Extends
Extends
<<Interface>>
Controller
+ execute(): AbstractMethod
+ set_state(self, state): void
+ update_state(self): AbstractMethod
+ transition(self): AbstractMethod
<<Class>>
SetDifficulty
+ render(self, **kwargs): String
+ assign(self, choice): String
Extends
<<Class>>
DisplaySlots
+ render(self, **kwargs): String
Extends
<<Class>>
SlotsView
Inherits
<<Interface>>
View
+ render(): String
1..*
Extends
<<Class>>
AskBet
+ render(self, **kwargs): String
+ validate_input(self, bet): Boolean
+ check_constraint(self, as_float, **kwargs): Float
<<Class>>
DataBaseConnection
+ url: string
+ db_user: string
+ password: string
+ create(): Type
+ create() : bool
+ retrieve(**kwargs) : string
+ update(**kwargs) : bool
+ delete(**kwargs) : bool
<<Abstract Class>>
Model
+ create(self): AbstractMethod
+ update(self): AbstractMethod
+ retrieve(self): AbstractMethod
+ delete(self): AbstractMethod
1..*
extend
<<Class>>
RandomUtils
+ getRandNum(): double
<<Class>>
UserModel
+ create_migration(self): void
+ migrate(self): void
+ validate(self, kwargs): String
+ create(self, kwargs): void
+ update(self, filter_kwargs, update_value): void
+ retrieve(self, kwargs): List
+ delete(self, kwargs): Boolean
+ check_if_id_exist(self, user_id): Boolean
<<Class>>
BlackJackDataModel
+ id: string
+ user_id: string
+ win_times: int
+ bust_times: int
+ BlackJackDataModel(id, user_id, win_times, bust_times): void
<<Class>>
UserDataModel
+ user_id: string
+ user_password: string
+ user_wallet: Float
<<Class>>
BlackJackModel
connectoin(self, db_conn): void
+ create_migration(self, db_conn): void
+ migrate(self): void
+ validate(self, **kwargs): void
+ create(self, **kwargs): void
+ update(self, **kwargs): void
+ retrieve(self, **kwargs): void
+ delete(self, **kwargs): void
<<Class>>
SlotsDataMode
+ id: string
+ user_id: string
+ highest_score:int
+ SlotsDataMode(id, user_id, highest_score): void
<<Class>>
RouletteDataModel
+ id: string
+ user_id: string
+ highest_score: int
+ RouletteModel(id, user_id, highest_score): void
<<Class>>
SlotsModel
+ field: type
+ method(): Type
<<Class>>
RouletteModel
+ field: type
+ method(): Type
Original draft plans for the project (not part of the UML)
Model Details
UserModel
RoulleteModle
SlotMachineModel
BlackJackModel
View Details
LoginView
UserRegistrationView
MenuView
RouletteView
SlotMachineView
BlackJackView
Controller Details
LogInController
MenuController
RouletteController
SlotMachineController
BackJackController


Removed from design.
Not yet implemented
Completely new implementation.
MVC Pattern
Strategy (typically these are Util classes/functions)
State
Factory
Command
Delegation
Prototype
Original draft plans for the project (this is not part of the UML)
Model Details
UserModel
RouletteModle
SlotMachineModel
BlackJackModel
View Details
LoginView
UserRegistrationView
MenuView
RouletteView
SlotMachineView
BlackJackView
Controller Details
LoginController
MenuController
RouletteController
SlotMachineController
BackJackController