Project 6 – Terminal Casino

Team Members:

Jay Seabrum

Dananjay Srinivas

Weitung Liao

Work Done:

Jay Seabrum: Completed implementing SlotsController and SlotsView. SlotsModel is rudimentarily set up.

Dananjay Srinivas: Set up the structure for BlackjackController, BlackjackView, BlackjackModel, and the abstract Controller and View classes.

Weitung Liao: Set up the structure for the abstract Model class and the database client file.

Changes/Issues Encountered:

So far there have been no major roadblocks in implementing what we have done so far. The only possible issue might be in making a GUI or keeping everything strictly in the terminal.

Patterns:

So far, the following patterns have been implemented:

Command: In the SlotsController, the controller asks for input form the user to set the difficulty level in slots. The controller itself doesn't know what to do with the input, but it delegates the interpretation of the input to the SlotsView AskBet module which houses the command pattern to assign the correct difficulty level for Slots.

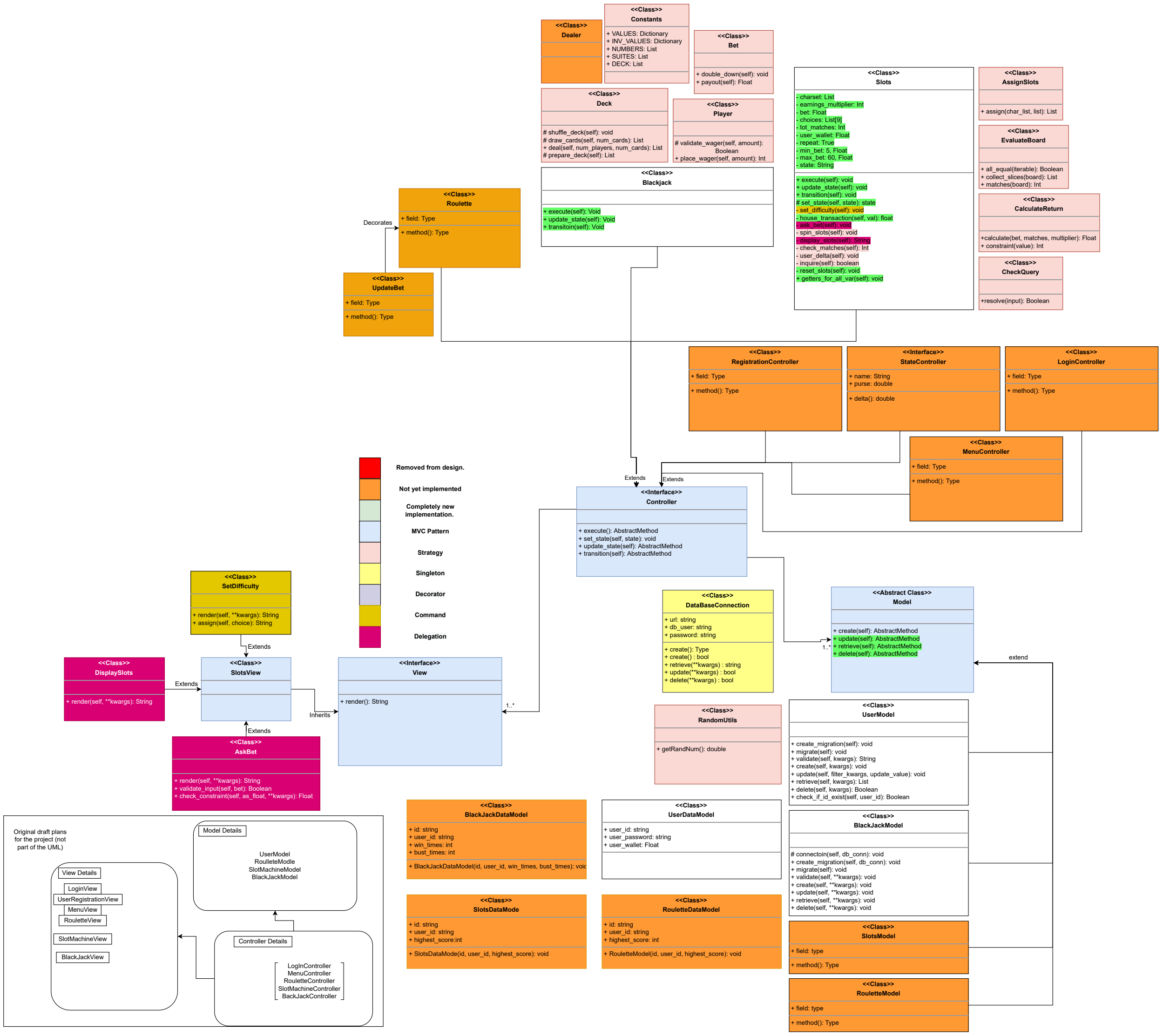
Strategy: In the SlotsController, for example, all of the modules delegate to another class somewhere else in the structure of the project. Because of this use of strategy, most of what the Slots controller does is to get things going in the right order.

Singleton: The DataBaseConnection class is a Singleton. We don't need multiple db connections for persisting data so having it invoked once is useful.

MVC: The backbone of the project is on this pattern. The games themselves are their own controllers, which then call specific Views to print useful information to the console or to assign useful returns to the controller. The Model's outline is set up to support specific data for each of the games and to handle the User so that the controllers can call these and process and change data as needed.

Plan for Next Iteration:

1. Create the MainManu, Settings, Login controllers.
2. Set up the structure for how users can log in and set passwords (and how these data need to be persisted)
3. Create a GUI at the end to ensure that emojis and other special characters display without worrying if computer's terminals can display them
4. Consider combining UserDataObject(Class describes the column in the user table in DB) with UserModel(Class handles the CRUD of the user table) by putting the column defining info as parameters of the UserModel. It would make it more neat while still have a good OO design
5. Finish the rest of the CRUD features for each data table we have
6. Consider making the DB connection as a static function instead of a singleton. It may make it just simpler?


<<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>>
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>>
Roulette
+ field: Type
+ method(): Type
Decorates
<<Class>>
UpdateBet
+ field: Type
+ method(): Type
<<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
Extends
<<Class>>
AskBet
+ render(self, **kwargs): String
+ validate_input(self, bet): Boolean
+ check_constraint(self, as_float, **kwargs): Float
<<Interface>>
View
+ render(): String
1..*
<<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): voic
<<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