Introduction to Python Programming WiSe 25/26
Group 7
Abigail Major, Nils Maier, Yining Zhou

Game: ♠ ♦ Blackjack ♥ ♣

♠ Summary:

Our program is a simple desktop application for playing Blackjack. In this game, users will be able to play according to the basic rules of Blackjack. To participate, users must create an account, and the game will track their best scores and account balances. These statistics will be displayed on two separate leaderboards.

♦ Functional Modules:

1. User Management:
   The User Management module is responsible for handling the lifecycle of user accounts and communicating with the game to manage user data.
   The key class/module in this module is: Player.
2. Blackjack Simulator:
   The Blackjack Simulator allows users to play Blackjack on their desktop.
   The key classes/modules in this module are Game and Card.

♣ Role Distribution:

To maximize productivity and efficiency, the workload is divided into three sections:

1. Front-End Development:
   Abigail, our Front-End Developer, will focus on the visual aspects of the program, ensuring the user interface is intuitive and easy to use.
2. Back-End Development:
   Nils, our Back-End Developer, will focus on building the core game logic and user account management, creating an authentic Blackjack experience.
3. Test & Integration:
   Yining will focus on integrating the back-end and front-end components, as well as creating test cases to identify and resolve bugs in the code.

♥ Technical Details:

1. GUI-Framework:
   For developing a desktop application, there are several libraries to choose from, such as Tkinter, TTK, PyQt, and PySide. After discussions, we decided to use PySide 6 for the front-end development due to the following reasons: Compared to Tkinter and TTK, PySide 6 offers more pre-defined modules, classes, and functions, allowing for quicker development of an advanced GUI. PySide is very similar to PyQt but comes with a more open license.
2. User Management Database:
   We will use Pandas to create and manage a lightweight database for user accounts.
3. Testing Framework:
   For automated testing, we will use Pytest. It supports running multiple test cases in parallel, reducing execution time. Additionally, it allows us to skip specific test methods during execution.

♠ Examples of Modules, Classes, and Functions:

The Game module is the core of the program. It functions like the Dungeon Master (DM) in a Dungeons & Dragons session—managing players and the dealer, controlling the game's progress, and tracking changes.
In our design, a game consists of 1 Player, associated with a user account and 1 Dealer, a special type of player who only exists during a game round.

The game's lifecycle is divided into 4 phases based on the game's progress and actions performed at each stage. This approach ensures clear visibility on when certain actions are enabled or disabled, and which UI elements should be shown or hidden.

♦ Phases:

1. **Phase 0: Starting Phase**
   The game is initialized with predefined values, and cards are dealt to both the dealer and the player. The only action available to the user at this stage is to place a bet.
2. **Phase 1: Player Action**
   In this phase, the player can either hit (draw another card) or stand (end their turn). If the player's hand exceeds 21 points (busts), this phase ends immediately.
3. **Phase 2: Dealer Action**
   The dealer will act based on the player's status and hand total. If the player busts, the dealer reveals their second card and stands. If the player stands with a hand total less than 21, the dealer will continue drawing cards until their total exceeds the player's or reaches 21, after which they will stand. If the dealer busts, the game proceeds to the next phase.
4. **Phase 3: Post-Game**
   The game determines the winner based on the players' and dealer's hand totals. If the player wins, they receive double their bet amount, and the game checks whether the player's balance has reached a new high score. If the player's balance is 0, their name is added to the "Hall of Shame," a section on the leaderboard. The game then prompts the user to start another round or return to the main menu.

♣ Key Functions in the Game Module:

1. calculate_hand(self, hand_owner)
   This function calculates the total value of a player's or dealer's hand. A key consideration is that the Ace (A) can be counted as either 1 or 11 points, depending on the other cards in the hand. The function ensures that the correct value is returned, accounting for any Aces.
2. calc_winner(self)
   This function determines the winner based on the final hand values of the player and dealer. There are four outcomes:
   - Player Busts, Dealer Stands: The dealer wins.
   - Both Player and Dealer Bust or Player and Dealer stands with same total smaller than 21: It's a tie (push).
   - Player Stands ($x < 21$), Dealer ($x<y \leq 21$): The dealer.
   - Player Stands ($x < 21$), Dealer ($y < x$): The player wins.

Appendix A: Table of Backend Classes, Attributes and Functions

Class	Attributes	Functions
Card	rank suit	

Player	name
	password
	score
	best_score
		hash_password()
		check_password()

Game	deck
	dealer
	dealer.hand
	dealer.status
	player
	player.hand
	player.status
	bet
	ai_play
	phase
		create_deck()
		shuffle_deck()
		draw_card()
		reset_round()
		initialize_game()
		start_game()
		deal_initial_hands()
		place_bet()
		add_bet()
		minus_bet()
		add_on_click()
		player_stands()
		dealer_draw()
		toggle_ai()
		ai_plays()
		calculate_hand()
		is_bust()
		phase_up()
		update_status()
		calc_winner()
		login_user()
		logout_user()
		save_score()
		load_all_scores()

login_panda (Module)		hash_password()
		create_file_check()
		load_df()
		user_exists()
		create_user()
		verify_user()
		get_score()
		set_score()
		list_scores()

Appendix B: Table of Frontend Classes, Attributes and Functions

Class	Attributes	Functions

MainWindow
(extends QMainWindow)
	WINDOW_FIXED_WIDTH
	WINDOW_FIXED_HEIGHT
	TITLE
	pages
		open_login_view()
		open_scoreboard_view()
		open_rules_view()
		open_place_bet_view()
		open_game_view()
		exit_game_view()

GameTable
(extends QWidget)
	dealer_card_view
	player_card_view
	v_layout
	hit_button
	stand_button
		disable_hit_button()
		disable_stand_button()
		open_new_game_dialog()

PlayerHandWidget (extends QWidget)	HAND_FIXED_WIDTH HAND_FIXED_HEIGHT hand_owner	render_cards()
CardView (extends QGraphicsView)	VIEW_WIDTH VIEW_HEIGHT is_revealed	update_view()

Menu
(extends QWidget)
	open_login_signal
	open_scoreboard_signal
	open_placebet_signal
	open_gametable_signal
	layout
		open_login_view()
		open_scoreboard_view()
		open_rules_view()
		open_place_bet_view()
		open_game_view()
		exit_game()

Login (extends QWidget)	username_input_field username_label password_input_field password_label signin_button layout	validate_signin check_username check_password check_username_exists check_correct_credentials trigger_account_creation
ApproveDialog (extends QDialog)	Layout title dialog_buttons	set_dialog_message(message)