Introduction

The firm 'Bring-it-on' has requested a Java or Python version of the game TwentyOne, a Dutch variation of Blackjack. Due to time restrictions, the firm is less concerned with the UI experience and more concerned with the back end design. Bring-it-on anticipates clean, understandable code with great architecture. Additionally, the program should include tests to ensure that everything is operational.

A Python implementation employing Object Oriented Programming was judged to be the best line of action to overcome this challenge. This was chosen because of its high level language, which allows for easy readability. Object-oriented programming helps Bring It On to comprehend the links between different components of the game and is efficiently scalable.

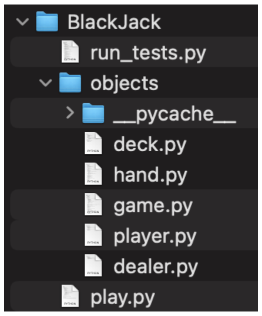
For the chosen UI, a command line interface was chosen, where the user may still have a fairly acceptable experience.

Important to note

The game's execution fully adheres to the rules specified in the assessment. However, it was decided to replace the terminology 'TwentyOne' to 'BlackJack' throughout the literature and code. To match the client's wishes, all instances of 'BlackJack' should be converted to 'TwentyOne' for future advancement.

What to expect after unzipping the document

After unzipping the folder, you will see the following:



```
BlackJack
    run_tests.py
    objects
        __pycache__
        deck.py
        hand.py
        game.py
        player.py
        dealer.py
    play.py
```

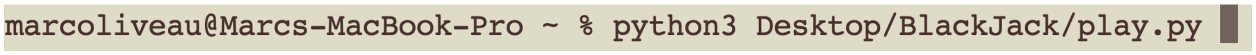
The objects folder contains the scripts with the different classes. Each script is named after the class that is created within the script. For example, the Hand class properties and functions can be found in the script hand.py.

The script run_tests.py contains all the unit tests for all of the classes functions.

The script play.py is the script that should be run to play the game.

How to run the game

Open a new terminal window, and run the script play.py. Make sure to use python3, as there are some features that python2 does not support.

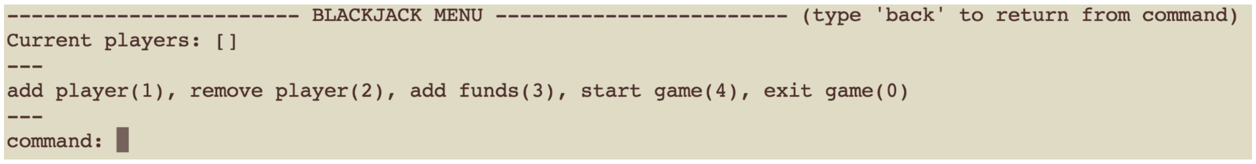


```
marcoliveau@Marcs-MacBook-Pro ~ % python3 Desktop/BlackJack/play.py
```

Make sure to specify the correct path for your device.

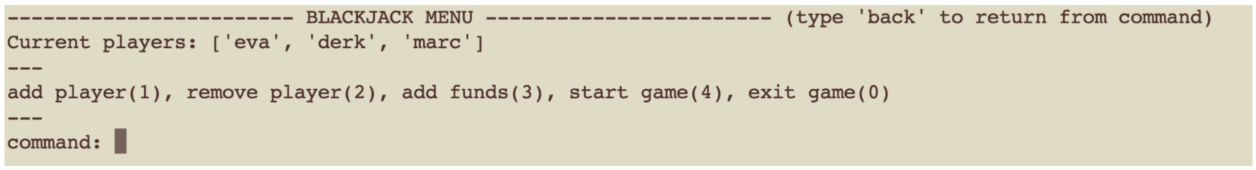
User Interface

The following display should appear after running the script



```
------------------------ BLACKJACK MENU ------------------------ (type 'back' to return from command)
Current players: []
---
add player(1), remove player(2), add funds(3), start game(4), exit game(0)
---
command:
```

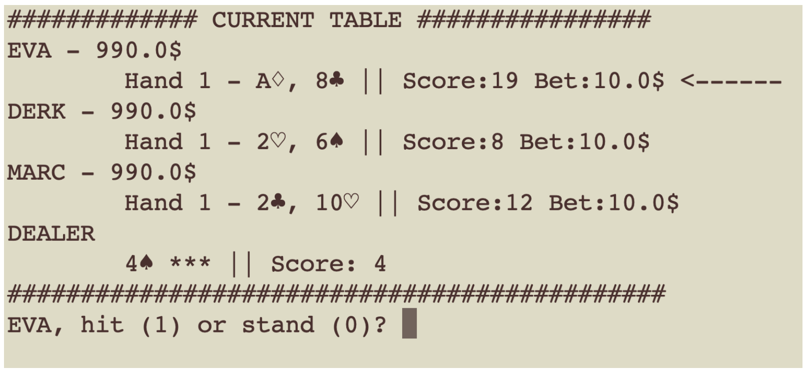
As seen, there is a menu displaying the current players in the game and some commands we can type. The number between the parentheses shows what character to type to run the given command.



```
------------------------ BLACKJACK MENU ------------------------ (type 'back' to return from command)
Current players: ['eva', 'derk', 'marc']
---
add player(1), remove player(2), add funds(3), start game(4), exit game(0)
---
command:
```

By typing ‘1’ and pressing ‘enter’, a player can be added to current players. This was done 3 times to get the current players seen above.

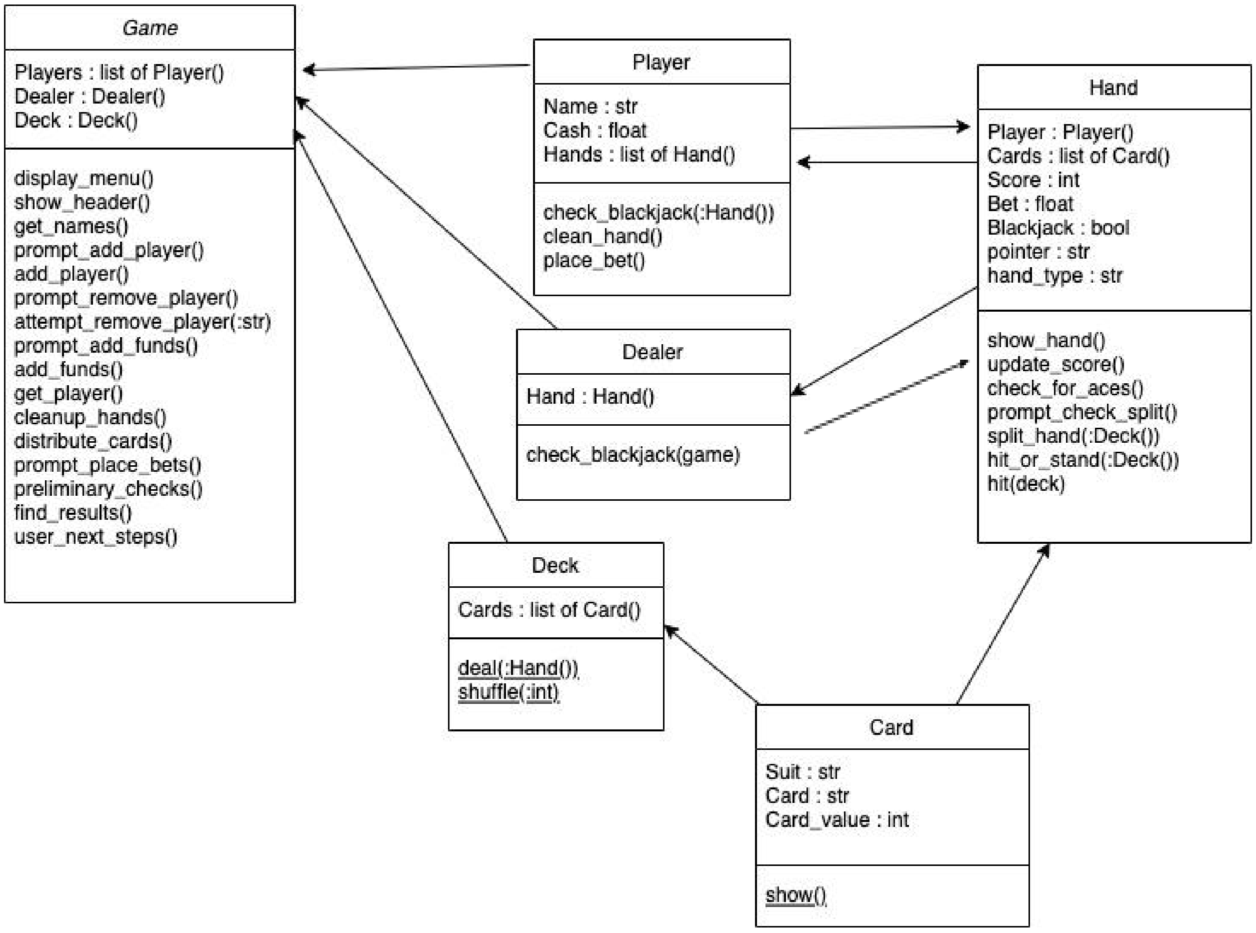
The UI is straight forward. The only place where there is a possibility of confusion is whose turn it is. This is indicated by the arrow found on the right of the hands. In this instance, it would be Eva’s turn.



```
############# CURRENT TABLE ################
EVA - 990.0$
        Hand 1 - A♢, 8♣ || Score:19 Bet:10.0$ <------
DERK - 990.0$
        Hand 1 - 2♡, 6♠ || Score:8 Bet:10.0$
MARC - 990.0$
        Hand 1 - 2♣, 10♡ || Score:12 Bet:10.0$
DEALER
        4♠ *** || Score: 4
############################################
EVA, hit (1) or stand (0)?
```

One thing to note is that whthe prompt “Place a bet:” is displayed, the user can’t C-c out of the script. To exit the game, enter a number and then when on another prompt type C-c.

Class Diagram



The class diagram is based on a java class diagram framework. Each box represents a separate class, and the attributes are presented with their respective data type under the class title. A list of the functions accessible in the specified class is presented beneath the attributes. If a function accepts input, the type of input is shown following a ':' within the parentheses. The arrows are only used to indicate when a given class is a property of another class.

Logical Flow of Game

Because the game is straightforward, constructing a pseudocode of the play.py script seems preferable than a flowchart. The script is also simple to understand, with a high degree of abstraction and only 40 lines of code. If any of the pseudocode does not make sense, I recommend reading it.

Pseudocode:

```
While user wants to keep playing
    Initialize all hands
    Shuffle deck
    Distribute a single card
    All users place a bet
    Distribute second card
    If dealer has blackjack
        Start new round
    If hand is splittable
        Allow user to split deck
    If hand is blackjack
        Hand is done for round
    All players hit or stand
    Dealer plays
    Update player cash
    Ask user if they want to keep playing
```

Unit Tests

All unit tests exist in the run_tests.py script. For this game, the library unit test is utilized. The tests also evaluate the edge cases and features. Debugging of user input occurs within the classes themselves. There are no unit tests for user input; only underlying logic is tested.

To make sure all tests work, run the script run_tests.py with python3

Areas for future improvement

Quick Fixes

- Clean up the script find_results.

  This function is a little bit messy, could use some abstraction and logical restructuring to make it more fluid
- Link Deck and Hand classes

  This would be nice to prevent passing inputs to certain functions that could be avoided. Also, it is more elegant both intuitively and for scalability.
- Force unique player names

  Currently, there is no issue with duplicated player names, however could lead to problems if we were storing historical data of users.

Next Steps

- Connect an SQL database to game

  This would enable the storage of user attributes. Currently, when the script is stopped, the user's data is lost. When a player is deleted or disconnects, their properties should be saved in a SQL server. When the player attempts to join, a new Player() instance is generated using the server's attributes. The username of the gamer would be the unique identity.
- Add more properties to Player class

  Connecting a database and keeping track of player statistics go hand in hand. Another item in the menu might be added to display certain users' history statistics, such as games played, average wager amount, number of wins, and so on. This would keep gamers engaged for a longer period of time.
- Create a Rest API framework

  A Rest API allows a player to connect from other devices without having to run the script themselves. When a player requests to join, their data is imported from the database and they can play using their prior cash. At the moment, all players are given $1,000 in cash. This might enable several devices to play the same blackjack game.

These features have not been implemented due to time restrictions, however the code is constructed in a way that permits these features to be readily added.