**SW Engineering CSC648**

**Blackjack Game development**

**Section 01 Team 05**

**10/19/22**

Seng Maw **–** Team leader & Scrum Master

Pritam Gautam **–** Frontend lead

Sneha Shrestha **–** Frontend lead

Lane Maimone **–** Backend lead

Nathaniel Miller **–** Backend lead

Tyler Hsieh **–** Backend lead & GitHub Master

Objective

**1. Data Definitions V2**

This should be reasonably consistent with Milestone 1 but should be expanded as needed and refined as per instructors’ feedback. Major data items that comprise of sub-data items have to be defined in full (list all its sub-data items, and for images/video list formats, max size etc.). **You must use all the data definitions and names consistently in all documents and SW, including UI text, naming for main variables, classes and database elements etc.** Focus on data items unique and important to your implementation. Be sure to cover ALL items critical to your project and especially those providing a competitive advantage. At this stage data describing user privileges, and main info (raw data, metadata, supporting data) have to be fully defined (as much as it is possible at this stage)

1. Object structure/class for cards:

a. structure/class will have variables for number attributes depending on the card, etc: King has a value of 10 and 4 has a value of 4. Ace will have a value of 1 or 11 and follow an if/else tree to handle the different scores that could be associated with an ace.

b. Card class may also have color variables and face variables for display purposes.

c. Card values is int.

d. cardStatus is Int variable for if the card is in play, discard, or in the deck.

e. All other variables are string.

2. Stack/Array of card objects with a max size that is a multiple of 52, depending on game mode, etc: Deck[52], Deck[104]

a. Deck.deal() pops top of Deck and moves popped card object to player

b. Deck.shuffle() randomly shuffles cards in stack

c. Deck.initialize() generates cards in a preset order

3. int turnNum to count turn

4. Object structure/class for players/dealer:

the player struct/class will be the parent of the dealer.

a. player has: vector<card> Hand[] for cards in hand,

b. int totalValue for total value of Hand[],

c. int Score for player's Score that's stored in database,

d. int playerStatus for if player is ingame, or out of the game.

e. int Credits for player credits.

f. the dealer does not have credits or score, and is an AI.

g. Player.hit() calls dealer to deal card to Player calling hit()

h. Player.pass() player no longer can hit, waits until everyone else finishes turn

i. Dealer.pass() like Player.pass() but also compares score with remaining players

5. **Global Functions**:

a. endTurn() ends global turn, increments int turnNum

b. startGame() begins game by initializing deck

c. int checkScore(Player) returns score of player

d. endGame() ends game and declares winner, also changes player credits/score

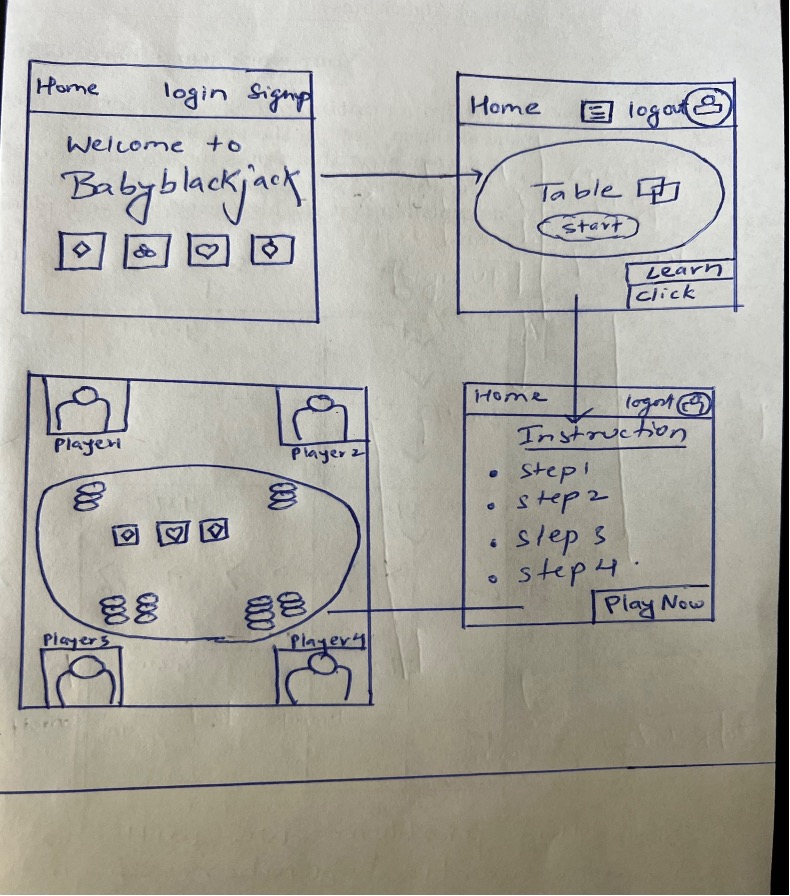
e. playerOut(Player) called by int checkScore when score is over 21

f. updatePlayerRank()

**2. Functional Requirements**

1. Create Account: 1
   1. Register Email
   2. Set Password
   3. Set Region
2. Join Game: 1
   1. Pick Single Player or Multiplayer
   2. Invite Registered Friends
3. Difficulty mode: 2
   1. Pick from Easy, Medium, Hard
4. Start/End Round of Blackjack: 1
5. Shuffle/Split Deck:1
6. Update Leaderboard: 2
   1. Local leaderboard
   2. Global leaderboard

**1.** **UI Mockups and StoryBoards**

****

# 

# 

# 

# **4. High level Architecture, Database Organization**

The below attributes of the game will be stored in the database.

|  |
| --- |
| **User Credentials** |
| userName Varchar (255) |
| userPassword Varchar (255) |

|  |
| --- |
| **Player Credentials** |
| playerScore Int |
| playerStatus Boolean |
| currentScore Int |

|  |
| --- |
| **LeaderBoard** |
| playerPosition Int |
| playerScore Int |

Add/ Delete/ Search Architecture

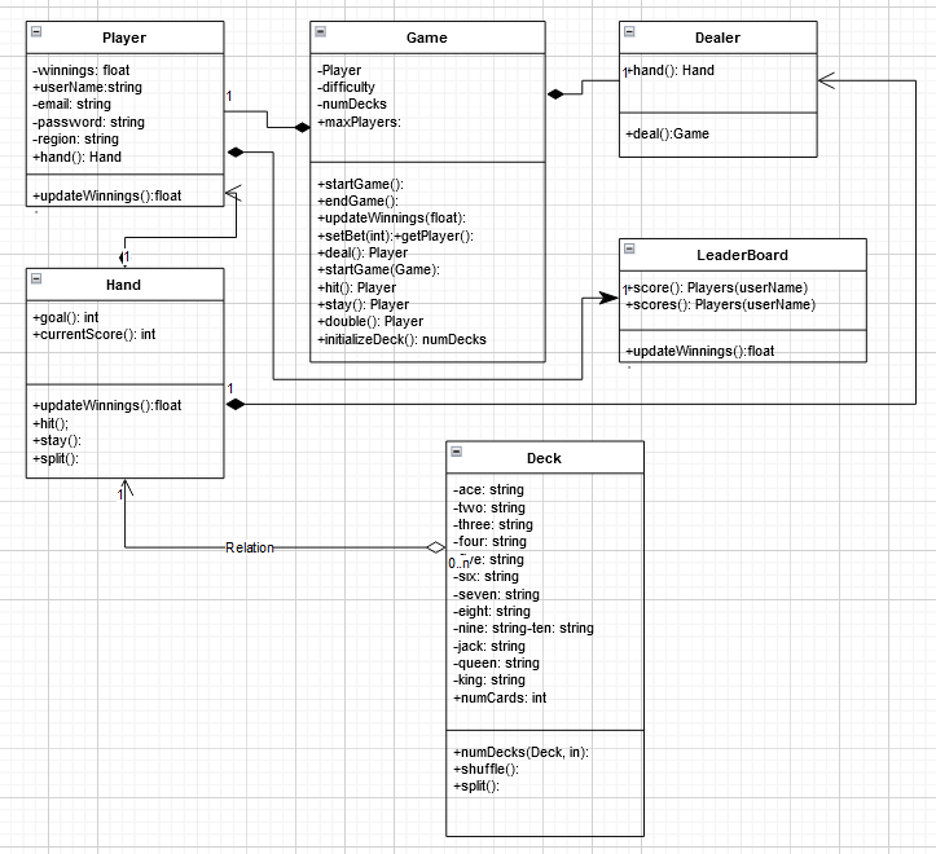
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | Add | Delete | Search | Display |
| User Credentials | Yes | Yes | No | Yes |
| Player Credentials | No | No | Yes | Yes |
| Leaderboard | No | No | Yes | Yes |

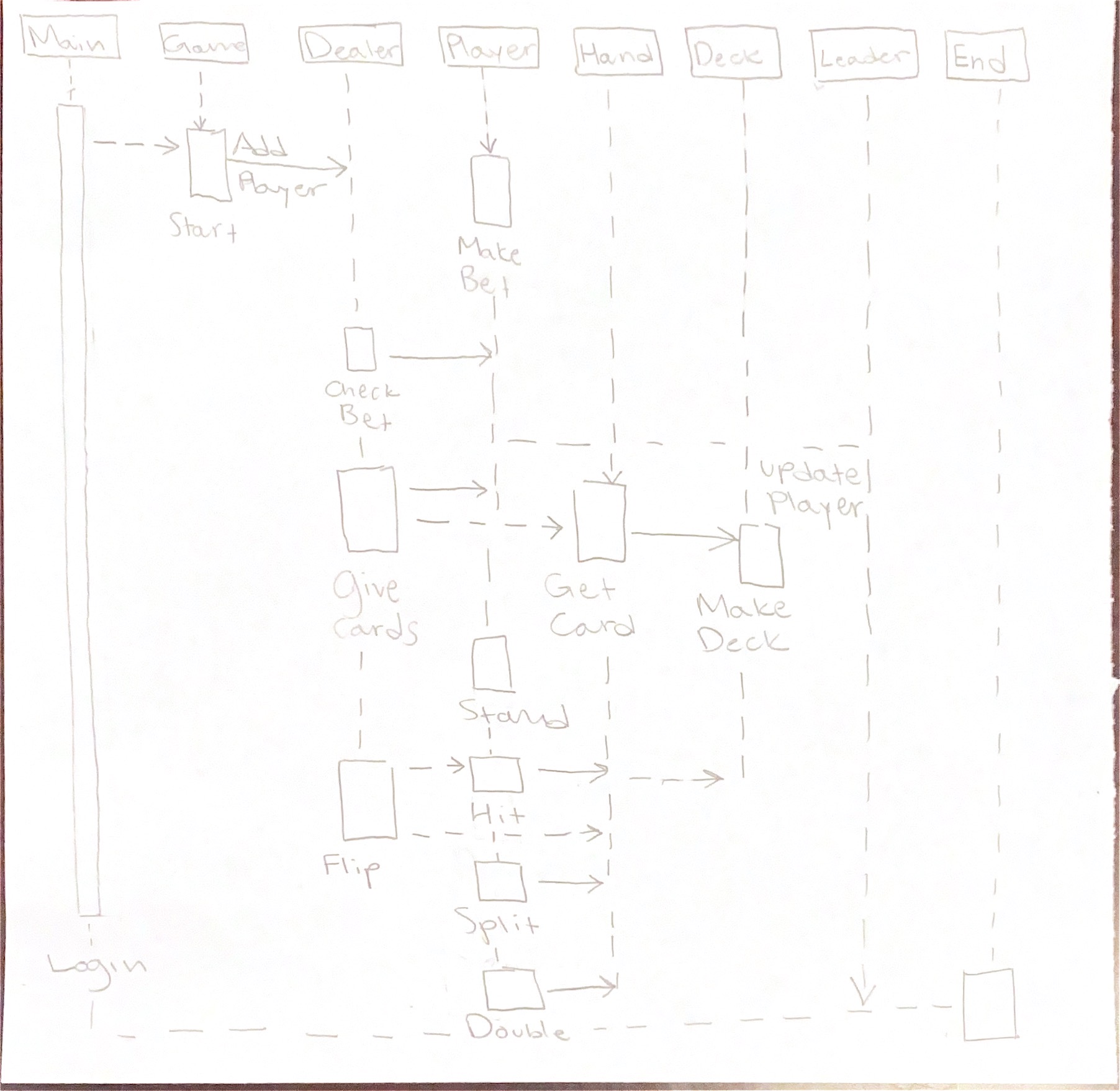
**DB operations**

The database operation for our game would be focusing on matching the stored credentials with the ones the users put in when logging into the web page using tokens and keys. Most data from our game would likely be the one that is collected during or after the game which is collected simultaneously would result to display after each game.

We are using React to create APIs for the project. In our case, the game’s functions require interaction with the database, which is where React comes in. Players looking to bet use their score as a currency. React connects this betting to their score from the database. Similarly, it works in the simpler cases such as for checking credentials when logging in or checking your place on the leaderboard.

**5. High Level UML Diagrams**



**

6. **Key risks**

**Progress on the finishing milestone**

- Not being able to carry out the skill or finish the study as planned

- The delay upon the mistakes or error we have faced while working on each part respectively

- The gap in linking between front end and back end

- Encountered difficulty with the chosen game engine

- SW prototype is not ready on time

**Solution** – We scheduled meetings, and met with the team members to resolve these issues. We updated our study plan accordingly and then we chose a new framework that is more suitable to work with the rest of the technologies we had chosen to work with.

**Challenges on the functionality of the project**

- The latency problem in browsing with mobile web browser

- The multiplayer function

Solution – We have scheduled the study plan and shared the tutorials and resources in the discord channel, Skill related risk will be resolved with planning the studies and working together when needed.

**7. Project management**

The task is divided into two main parts as Back-end and Front-end.It is focused as a group work according to the tasks.

We use discord as the main tool to manage the tasks and meetings.

We generally have two meetings in a week. However, for this milestone, we had more than two meetings. Team lead had different meetings with different sub-group such as (Back-end and Front-end ) along the process to make sure we were on the track. We resolve each other’s conflicts upon our tasks in these meetings and discuss the difficulties we have stumbled upon while working.

We had a serious group meeting relating to our choice of game engine because we have figured that the game engine was very recent and the resources to make it work with the rest of the chosen technologies aren’t available abundantly. After the meetings, we chose another game engine and got it to work.

All the pictures we used in the project are downloaded from the internet for educational purpose.

# 

# 