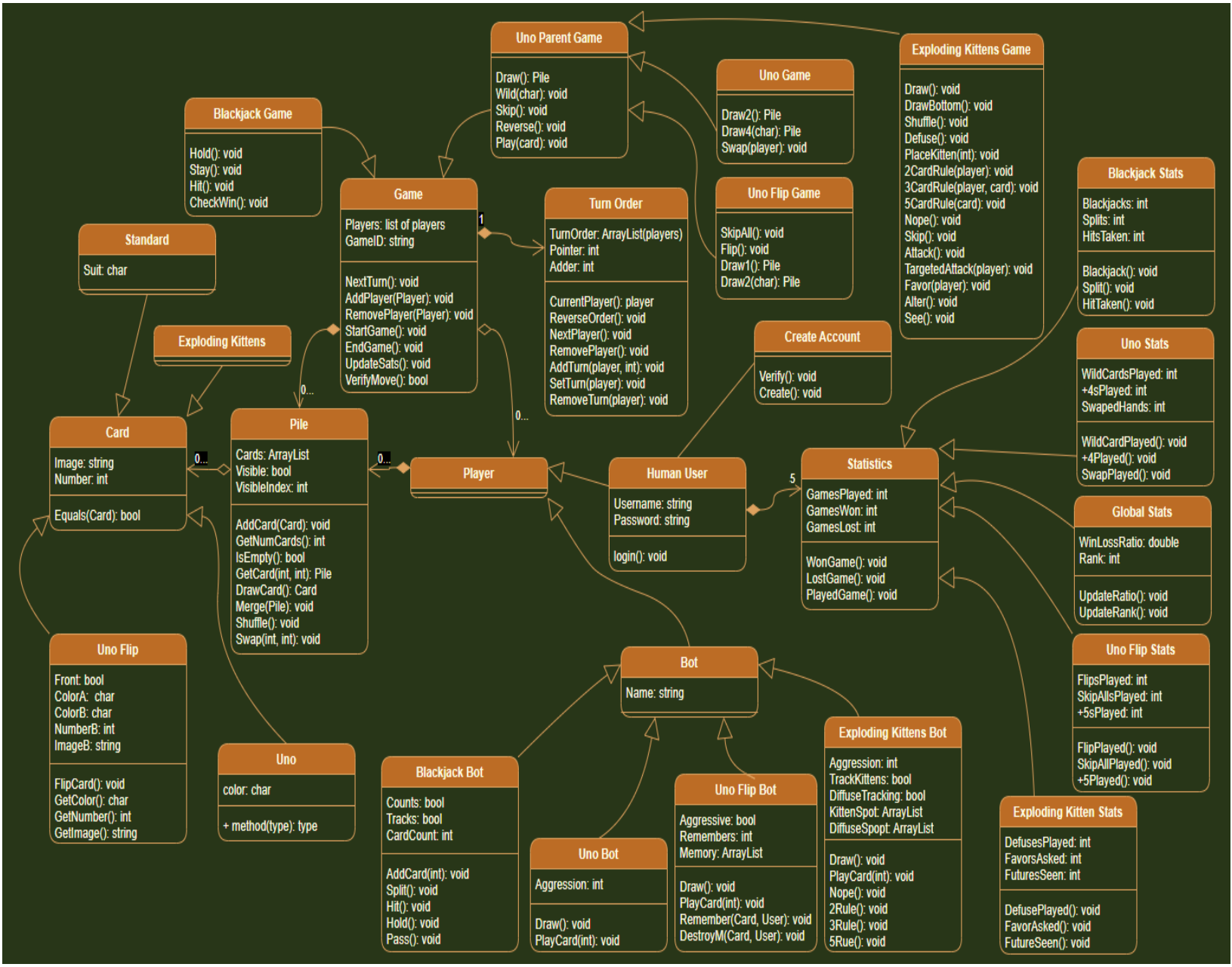
Textual Analysis

CS320-103: Software Engineering and Design
Due Date: March 12, 2022
Paul Walter , Ian Viveiros , Braden Fleming , Mark Williams

Methods Classes Attributes Unused Interfaces

Nouns	Verbs
User UserList Opponent Player Game ID Bot Statistics Turn Hand Card Deck Pile Color Number Image	Join Host Draw Play Swap Choose Skip Reverse Draw2 Draw4 Draw5 Drawuntilcolor Discard Hit Hold Freeze Split Shuffle Nope Flip


Uno Parent Game
Draw(): Pile
Wild(char): void
Skip(): void
Reverse(): void
Play(card): void
Uno Game
Draw2(): Pile
Draw4(char): Pile
Swap(player): void
Exploding Kittens Game
Draw(): void
DrawBottom(): void
Shuffle(): void
Defuse(): void
PlaceKitten(int): void
2CardRule(player): void
3CardRule(player, card): void
5CardRule(card): void
Nope(): void
Skip(): void
Attack(): void
TargetedAttack(player): void
Favor(player): void
Alter(): void
See(): void
Blackjack Game
Hold(): void
Stay(): void
Hit(): void
CheckWin(): void
Game
Players: list of players
GameID: string
NextTurn(): void
AddPlayer(Player): void
RemovePlayer(Player): void
StartGame(): void
EndGame(): void
UpdateSats(): void
VerifyMove(): bool
Turn Order
TurnOrder: ArrayList(players)
Pointer: int
Adder: int
CurrentPlayer(): player
ReverseOrder(): void
NextPlayer(): void
RemovePlayer(): void
AddTurn(player, int): void
SetTurn(player): void
RemoveTurn(player): void
Uno Flip Game
SkipAll(): void
Flip(): void
Draw1(): Pile
Draw2(char): Pile
Blackjack Stats
Blackjacks: int
Splits: int
HitsTaken: int
Blackjack(): void
Split(): void
HitTaken(): void
Standard
Suit: char
Exploding Kittens
Create Account
Verify(): void
Create(): void
Uno Stats
WildCardsPlayed: int
+4sPlayed: int
SwapedHands: int
WildCardPlayed(): void
+4Played(): void
SwapPlayed(): void
Card
Image: string
Number: int
Equals(Card): bool
Pile
Cards: ArrayList
Visible: bool
VisibleIndex: int
AddCard(Card): void
GetNumCards(): int
IsEmpty(): bool
GetCard(int, int): Pile
DrawCard(): Card
Merge(Pile): void
Shuffle(): void
Swap(int, int): void
Player
Human User
Username: string
Password: string
login(): void
Statistics
GamesPlayed: int
GamesWon: int
GamesLost: int
WonGame(): void
LostGame(): void
PlayedGame(): void
Global Stats
WinLossRatio: double
Rank: int
UpdateRatio(): void
UpdateRank(): void
Uno Flip
Front: bool
ColorA: char
ColorB: char
NumberB: int
ImageB: string
FlipCard(): void
GetColor(): char
GetNumber(): int
GetImage(): string
Bot
Name: string
Uno Flip Stats
FlipsPlayed: int
SkipAllsPlayed: int
+5sPlayed: int
FlipPlayed(): void
SkipAllPlayed(): void
+5Played(): void
Uno
color: char
+ method(type): type
Blackjack Bot
Counts: bool
Tracks: bool
CardCount: int
AddCard(int): void
Split(): void
Hit(): void
Hold(): void
Pass(): void
Uno Bot
Aggression: int
Draw(): void
PlayCard(int): void
Uno Flip Bot
Aggressive: bool
Remembers: int
Memory: ArrayList
Draw(): void
PlayCard(int): void
Remember(Card, User): void
DestroyM(Card, User): void
Exploding Kittens Bot
Aggression: int
TrackKittens: bool
DiffuseTracking: bool
KittenSpot: ArrayList
DiffuseSpopt: ArrayList
Draw(): void
PlayCard(int): void
Nope(): void
2Rule(): void
3Rule(): void
5Rue(): void
Exploding Kitten Stats
DefusesPlayed: int
FavorsAsked: int
FuturesSeen: int
DefusePlayed(): void
FavorAsked(): void
FutureSeen(): void
1
0...
0...
0...
0...
5

As we are creating a game repository, we as a team implemented fundamental classes for the functionality of the card games in our web application. For our basis, classes such as game, pile, statistic, player, and bot are necessities for the project. Each of which are inheriting methods from corresponding inheritances. The UML is divided into three sections primarily, those being the pile, player, game, and statistic class. One of the most important classes that is established in the UML is the player class which possesses no methods of its own but inherits both the human user interface and the bot interface. The player is considered both the human user and bot that is playing a game. Both possess the same rules, similar preconditions, and tasks but, they have different needs to join a game. The bot simply needs to possess a name to be able to join a game while the human user must either login, sign in as a guest or create an account. The human user will need to possess a username and password.

The pile class is dependent on the information about the player class which declares the necessary fields for the deck. Simply, the player knows about the pile. The player will know the number of cards they have possession of(arrayList), if a card is visible, and which card in their list is visible. The pile itself is then able to perform various actions such as being shuffled, merging a pile, and knowing when it is empty. The pile will also be able to tell when and how many cards are being added to the pile, which index needs to be swapped and whether the player is drawing a card from the pile. In order for a player to make a move such as this, we must have a class that determines whose turn it is; in our case, this is the game class.

The game class shares a relationship with the player. While the game class is dependent on the player class, the game can exist independently from the player. This is opposite for the pile because although the game class is dependent on the pile class, it cannot exist independently from it. The game class determines whether a move by the player is legal in regards to their turn or in terms of the pile. The main deck is also dependent and references to the type of card and number assigned to that card. The card class will inherit the methods and fields for each card game as a child class. Additionally, the game class will rely on and reference to the

Gaining a deeper understanding of our process in making the bot class and determining it is a fundamental class for our project, it was determined that the bot can be as applicable as the human user in regards to the start of a game. In a realistic match, we want the bot to have the same limitations with no disadvantages or advantages attached. In order to do this, we must have a parent class called bot and several child classes that inherit from the parent. Each child class possesses the specific actions that a player can perform in each game.

The final fundamental feature that will be present in our game repository is the statistics class. It will keep track of all of the games that are played and find the win/loss average, this includes unique features as well. In order to do so, we will need to gather and track these actions in the games. We have several card game statistic classes that will be inheritances to the parent class statistics. As an example, blackjack will count the number of blackjacks, hits and splits a player may perform but will also keep track of the number of times the player won and lost a game. As these statistics are tracked, they will be a composition of the human user and will ultimately be dependent on the statistics.