Eric Crandall Poker Face

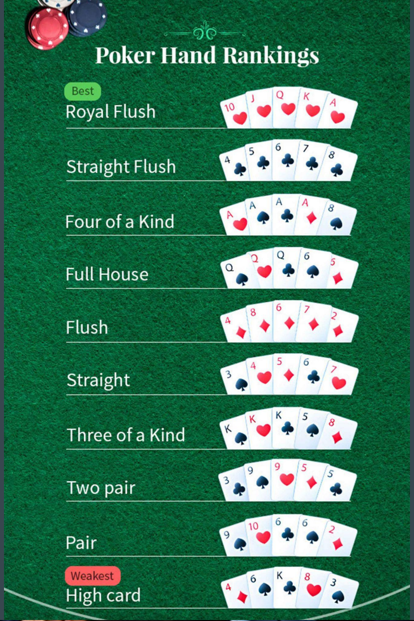
McEwan Bain, Jake VanZyverden, Gabriel Hoing

Project Overview

- Our Game: Texas Hold'Em Poker
  - With a South Park (and Dr. Crandall) theme!
- Our Main Features:
  - Automatic Scoring
  - Betting: Calls, Raises, Folds, etc
  - Standard Deck of 52 Playing Cards
    - Community and Individual Cards
- Limitations:
  - Time!!!!
  - Communicating
- Assumptions:
  - Players know how to play

Da Rules

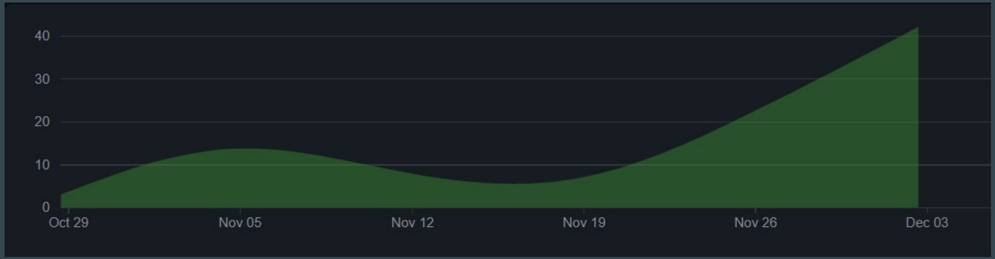
- Texas Hold'Em:
  - Each player has a 2 card hand
  - There are 5 community cards on the table
    - Start face down and are slowly revealed
  - Each player attempts to make the best hand possible out of their cards and the community cards
  - Each player is allowed to bet before the next set of cards are revealed
  - Players can choose to play x number of rounds or play until all but one player runs out of $



Project Requirements

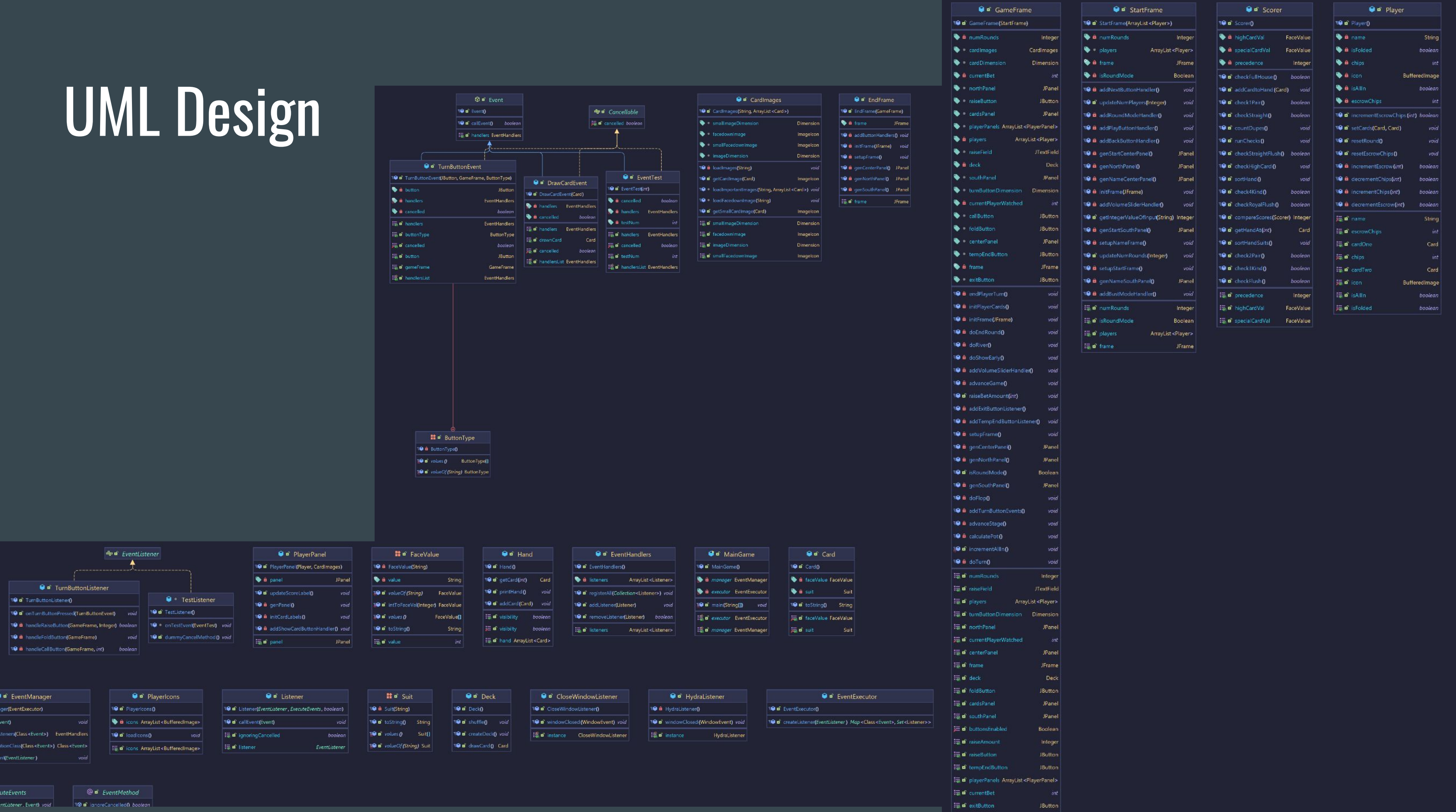
- Playable for between 2 and 7 players
  - Each player can set their name and are assigned an icon
- Scoring
  - Automatically create the best possible hand for each player
  - Compare the scores of each player and determine who has the best hand
- Betting
  - Players are able to bet x amount of chips on their turn
  - If someone has already bet, players can choose to raise or call the bet
  - Players are allowed to fold if they no longer wish to bet
- Sound Thread
  - A non-functional requirement
  - Through the use of multi-threading, our game constantly plays background music related to our theme

Project Solution Approach



- Backend First
  - Before any gui work, we wrote out all the classes we thought we would need
  - Tested with unit tests
- Then GUI
  - Once the backend was in a good state, we switched to the visual layout of the gui
  - Did not really work on functionality until after the gui was done
- Putting Everything Together
  - Once the gui and our framework was all laid out, then we started working on the game
  - This was relatively easy because all the tools we needed were already built
  - However it was also here where we dropped/simplified a lot of our ideas

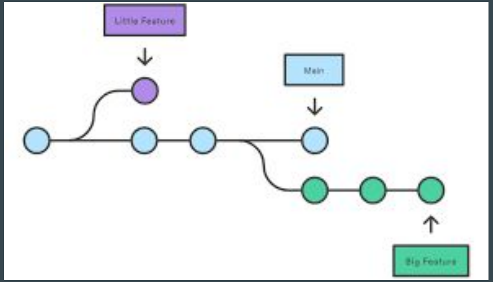
UML Design



Event
Cancellable
TurnButtonEvent
DrawCardEvent
EventTest
CardImages
EndFrame
ButtonType
GameFrame
StartFrame
Scorer
Player
EventListener
TurnButtonListener
TestListener
PlayerPanel
FaceValue
Hand
EventHandlers
MainGame
Card
EventManager
PlayerIcons
Listener
Suit
Deck
CloseWindowListener
HydraListener
EventExecutor
EventMethod

Team Collaboration Approaches

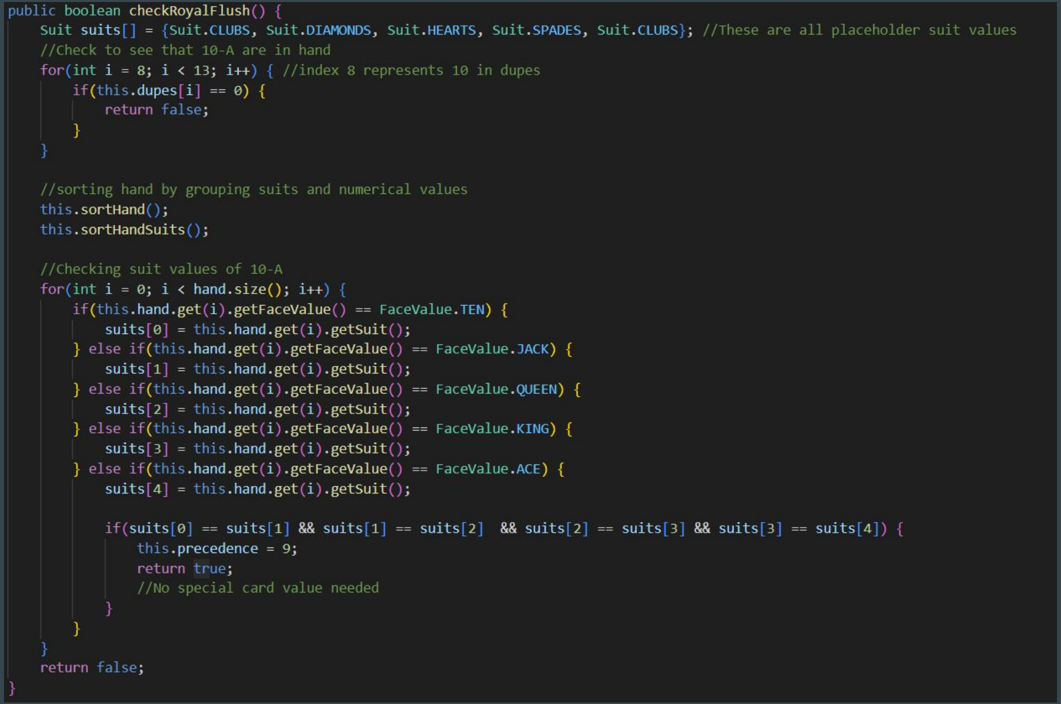
- Messaging:
  - Main form of communication was through text messaging
  - Clarified questions in person
- Work Split:
  - Nothing was 'assigned' to anyone
  - Messaged the group before starting something new, and after finishing
  - Tried our best to do a fair split of the workload
- GitHub:
  - BRANCHES!
    - Extremely useful for this project, saved us a lot of headache
    - Easy to see merge conflicts and sort out before causing major issues
- Group Coding:
  - Most of the work was done separately
  - Had 4 in person group meetings
    - 2 smaller coding sessions, 2 big crunches



Testing, Validation, and Acceptance Plan

- Unit Testing:
  - The big one!
  - For the backend we wrote loads of tests
    - Card, Deck, Scorer, etc
- Console Outputs:
  - For certain process like image loading, we tested by outputting messages to terminal
- Play Testing:
  - To test the gui and that everything was put together properly, we play tested!
- Integration Testing:
  - Pull Requests were reviewed before being merged
  - Reran all tests after merging to see if there were unexpected breaks

Testing Example

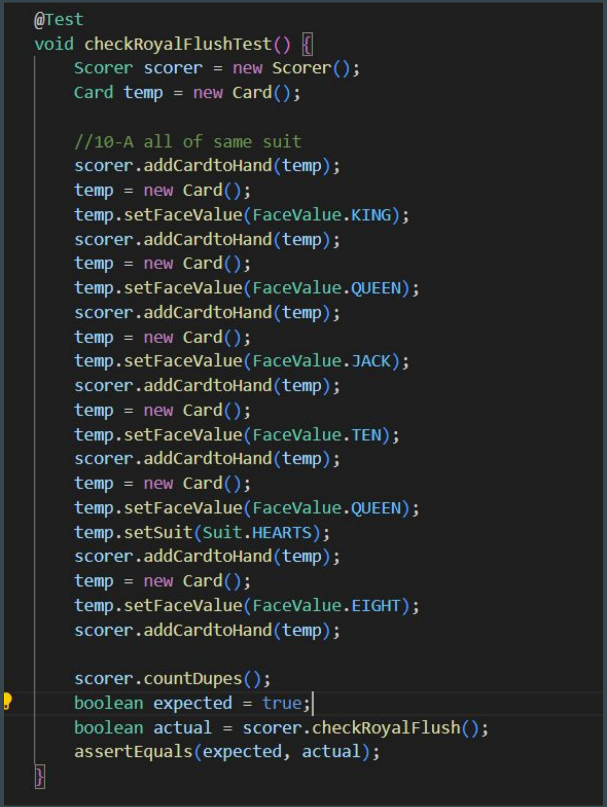


```java
public boolean checkRoyalFlush() {
    Suit suits[] = {Suit.CLUBS, Suit.DIAMONDS, Suit.HEARTS, Suit.SPADES, Suit.CLUBS}; //These are all placeholder suit values
    //Check to see that 10-A are in hand
    for(int i = 8; i < 13; i++) { //index 8 represents 10 in dupes
        if(this.dupes[i] == 0) {
            return false;
        }
    }

    //sorting hand by grouping suits and numerical values
    this.sortHand();
    this.sortHandSuits();

    //Checking suit values of 10-A
    for(int i = 0; i < hand.size(); i++) {
        if(this.hand.get(i).getFaceValue() == FaceValue.TEN) {
            suits[0] = this.hand.get(i).getSuit();
        } else if(this.hand.get(i).getFaceValue() == FaceValue.JACK) {
            suits[1] = this.hand.get(i).getSuit();
        } else if(this.hand.get(i).getFaceValue() == FaceValue.QUEEN) {
            suits[2] = this.hand.get(i).getSuit();
        } else if(this.hand.get(i).getFaceValue() == FaceValue.KING) {
            suits[3] = this.hand.get(i).getSuit();
        } else if(this.hand.get(i).getFaceValue() == FaceValue.ACE) {
            suits[4] = this.hand.get(i).getSuit();

            if(suits[0] == suits[1] && suits[1] == suits[2]  && suits[2] == suits[3] && suits[3] == suits[4]) {
                this.precedence = 9;
                return true;
                //No special card value needed
            }
        }
    }
    return false;
}
```



```java
@Test
void checkRoyalFlushTest() {
    Scorer scorer = new Scorer();
    Card temp = new Card();

    //10-A all of same suit
    scorer.addCardtoHand(temp);
    temp = new Card();
    temp.setFaceValue(FaceValue.KING);
    scorer.addCardtoHand(temp);
    temp = new Card();
    temp.setFaceValue(FaceValue.QUEEN);
    scorer.addCardtoHand(temp);
    temp = new Card();
    temp.setFaceValue(FaceValue.JACK);
    scorer.addCardtoHand(temp);
    temp = new Card();
    temp.setFaceValue(FaceValue.TEN);
    scorer.addCardtoHand(temp);
    temp = new Card();
    temp.setFaceValue(FaceValue.QUEEN);
    temp.setSuit(Suit.HEARTS);
    scorer.addCardtoHand(temp);
    temp = new Card();
    temp.setFaceValue(FaceValue.EIGHT);
    scorer.addCardtoHand(temp);

    scorer.countDupes();
    boolean expected = true;
    boolean actual = scorer.checkRoyalFlush();
    assertEquals(expected, actual);
}
```

Live Demo Time

Let’s Do it!

Summary

- A Fun and Anxiety Filled Ride!
- We shot for the moon...
    - We ended up dropping and simplifying some components in order to meet the deadline
    - While we didn't achieve all our initial goals, we're very proud of how things turned out
- Working as a Team
    - Frustrating at times
    - A great learning experience