

Projeto final

Truco

Objetivos

Esse projeto tem por objetivo a criação de uma aplicação para a simulação de um jogo de Truco, aplicando os conhecimentos obtidos no curso INF1900.

Arquitetura

Este projeto Truco foi desenvolvido utilizando linguagem de programação C++ e padrão de arquitetura MVC (Model, View, Controller).

O padrão MVC foi considerado porque:

- Proporciona agilidade na atualização da interface da aplicação;
- A manutenção do código se torna mais fácil;

Para o desenvolvimento da interface gráfica, MFC (Microsoft Foundation Classes) foi escolhido como biblioteca, por conta dos seguintes fatores:

- É orientado a objetos;
- Útil para a criação de interfaces de usuários mais complexas;
- Fornece wrapper que pode ser utilizado em grande parte das APIs Win32 e COM;

Além das escolhas de arquitetura, seguem alguns exemplos de implementações dos conceitos aprendidos em sala de aula que foram aplicados dentro do sistema:

Threads:

Segue o exemplo de utilização de threads na classe Round.cpp

```
void Round::StartPlayingThread()
{
    // Creates playing thread to handle the player actions
    playingThread = std::thread([this]() {
        std::unique_lock l(playingMutex);

        while (true)
        {
            playingConditionVariable.wait(l);

            this->PlayCard();
            this->NextPlayer();
        }
    });

    playingThread.detach();
}
```

Projeto – Truco

Mutex:

Exemplo de utilização de mutex na classe CPUPlayer.cpp

```
void CPUPlayer::MonitorRoundState(Round* round)
{
    // Bot player keeps monitoring the round in order to play when it is its turn
    std::thread activePlayerThread([this, round]() {
        std::unique_lock l(playerMutex);

        while (true)
        {
            AnalyzeCurrentBet(round);
            TriggerPlayIntention(round);

            // A CPU Thread acquires mutex lock until it waits
            // When it is waiting, mutex is unlocked and the next thread can execute loop
            playerConditionVariable.wait(l);
            // After the condition variable notify all threads, they wake up, but the first to do it acquires the lock and execute the loop
            // When it starts waiting again, the second thread will be able to acquire mutex lock and perform the loop
        }
    });

    activePlayerThread.detach();
}
```

Váriavel Condicional:

Exemplo de utilização de variavel condicional na classe CPUPlayer.cpp

```
void CPUPlayer::NotifyPlayers(bool notifyAll)
{
    if (notifyAll) {
        // Notifies all CPU Player threads for playing action
        playerConditionVariable.notify_all();
    }
    else {
        // Notifies only one CPU Player thread to randomly decide about Truco request
        playerConditionVariable.notify_one();
    }
}
```

Tratamento de Exeção:

Classe IndexOutOfRangeException.cpp, para exceção customizada.

```
class IndexOutOfRangeException : public std::runtime_error {
public:
    IndexOutOfRangeException(const std::string& message)
        : std::runtime_error(message) {}
};
```

E abaixo um exemplo de sua utilização na classe Player.cpp.

```
void Player::RemoveSelectedCard()
{
    if (selectedCardIndex < 0 || selectedCardIndex >= hand.size()) {
        throw IndexOutOfRangeException("Selected card index is out of range.");
    }

    Card cardToRemove = GetHand()[selectedCardIndex];
    auto it = std::find(hand.begin(), hand.end(), cardToRemove);
    if (it != hand.end()) {
        hand.erase(it);
    }
    selectedCardIndex = -1;
}
```

Classe PlayerCreationException.cpp, para exceção customizada.

```
class PlayerCreationException : public std::exception {
private:
    std::string message;
public:
    PlayerCreationException(const std::string& msg) : message(msg) {}

    virtual const char* what() const throw() {
        return message.c_str();
    }
};
```

E abaixo, um exemplo de sua utilização na classe TrucoController.cpp.

```
TrucoController::TrucoController()
{
    firstPlayer = 0;
    std::array<std::unique_ptr<Player>, 4> players;

    try {
        round = std::make_unique<Round>();

        // Criação de jogadores humanos e CPUs.
        players[0] = HumanPlayer::Create(L"HUMAN-Pedro");
        players[1] = CPUPlayer::Create(L"BOT-Priscila", round.get());
        players[2] = HumanPlayer::Create(L"HUMAN-Adnan");
        players[3] = CPUPlayer::Create(L"BOT-Danilo", round.get());

        round->SetPlayers(std::move(players));
        round->RoundOverEventListener(std::bind(&TrucoController::NotifyRoundOver, this));

        StartRoundHandlerThread();
    }
    catch (const std::exception& e) {
        throw PlayerCreationException("Failed to initialize TrucoController: " + std::string(e.what()));
    }
}
```

Smart Pointer:

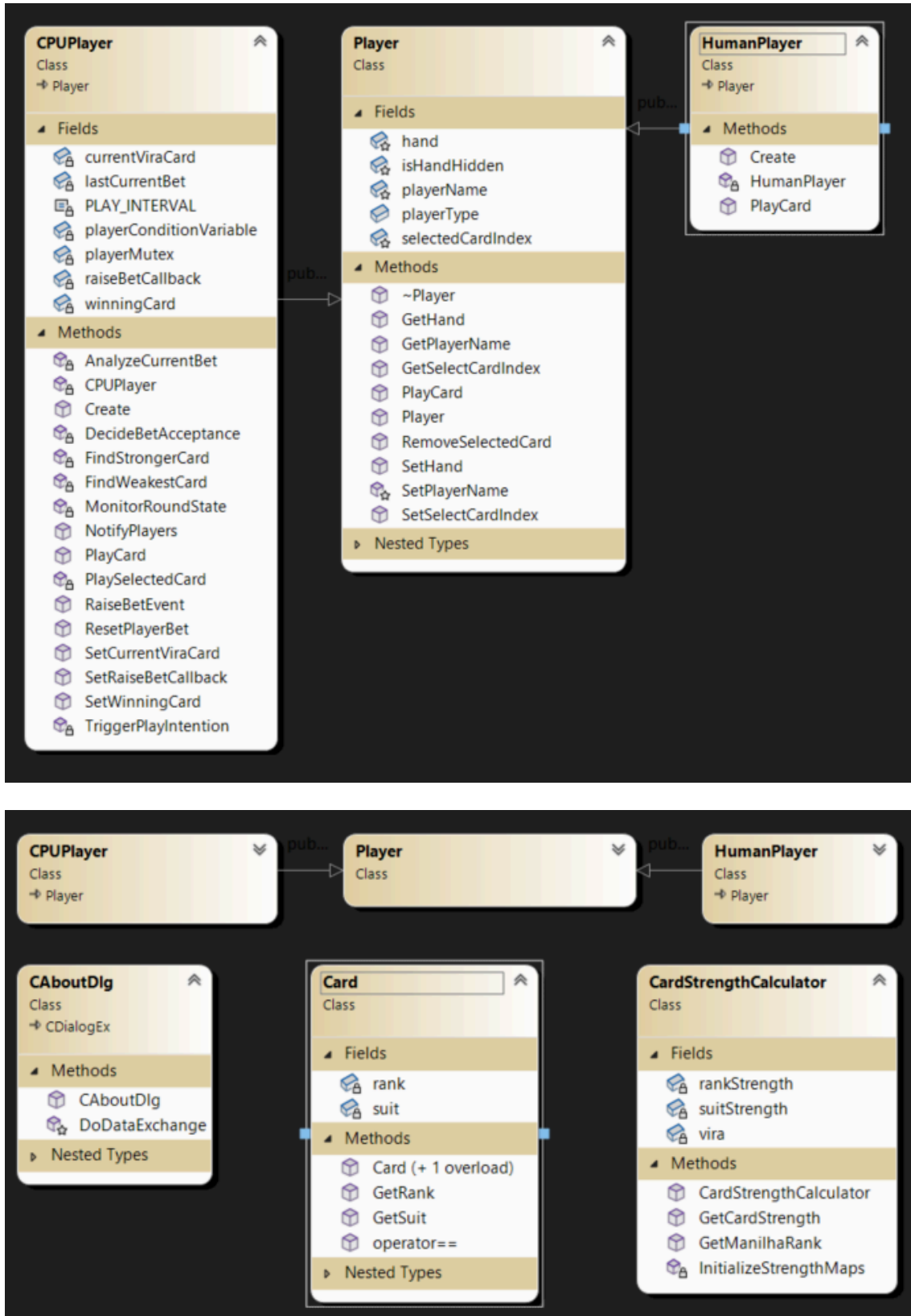
Exemplo de utilização de variavel condicional na classe CPUPlayer.cpp.

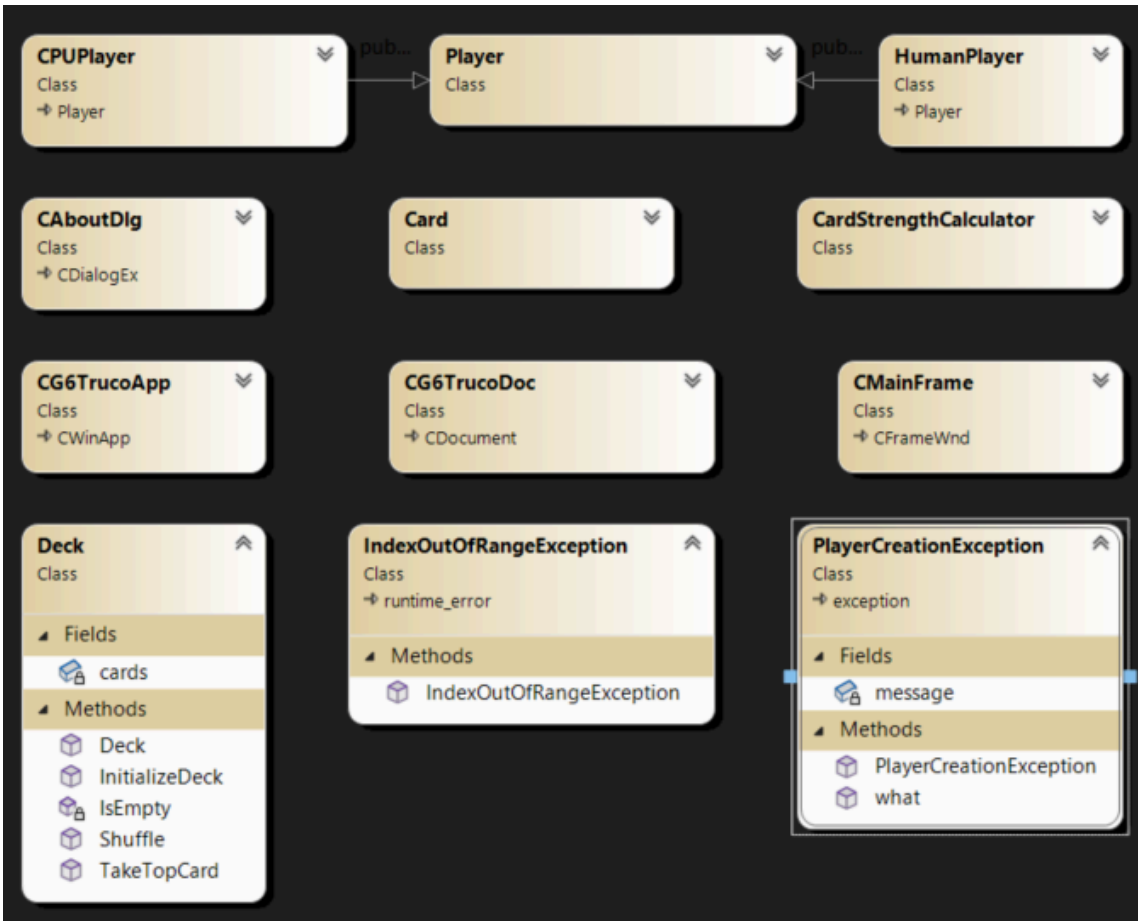
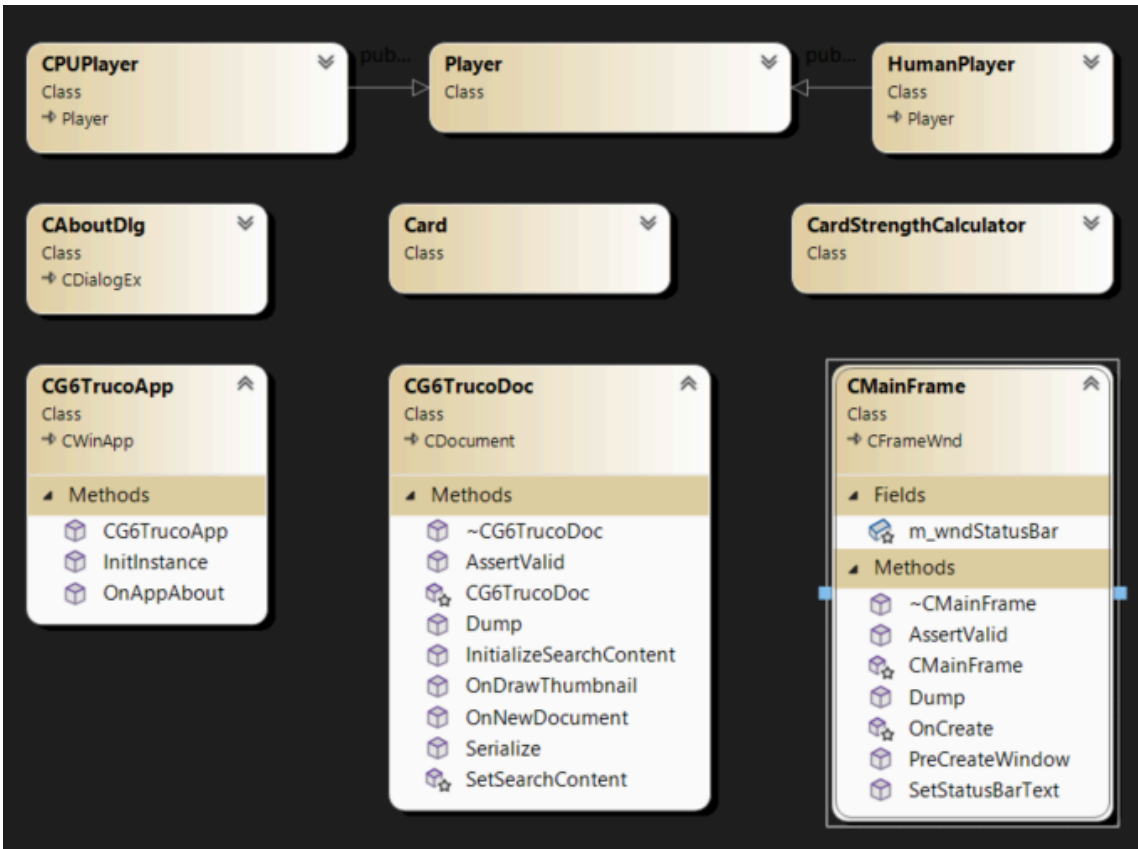
```
std::unique_ptr<CPUPlayer> CPUPlayer::Create(const CString& name, Round* round)
{
    std::unique_ptr<CPUPlayer> player = std::unique_ptr<CPUPlayer>(new CPUPlayer(name));
    player->MonitorRoundState(round);

    return std::move(player);
}
```

Diagrama de Classes

O diagrama de classes foi gerado automaticamente com o auxílio do Visual Studio 2022 Community.





Projeto – Truco

The screenshot displays the Xcode interface with three class view panels showing the hierarchy of the Truco project. The classes are organized into a tree structure, with each class panel showing its fields and methods.

Round Class

- Fields
 - activePlayerIndex
 - currentBet
 - currentTrucoCall
 - deck
 - isRoundOver
 - players
 - playingConditionVariable
 - playingMutex
 - playingThread
 - points
 - possibleBets
 - roundCards
 - roundInformationChangedEvent
 - roundOverEvent
 - roundWinnerTeam
 - vira
 - winningCard
- Methods
 - DealCardsToPlayers
 - DefineWinningCard
 - DenyBet
 - DetermineRoundPoint
 - GetActivePlayer
 - GetActivePlayerIndex
 - GetAllPlayers
 - GetCPUActivePlayer
 - GetCurrentBet
 - GetCurrentTrucoCall
 - GetNextBetValue
 - GetPoints
 - GetViraCard
 - GetWinnerTeam
 - GetWinningCard
 - IsHumanPlayer (+ 1 overload)
 - IsRoundOver
 - NewCardIsStronger
 - NextBet
 - NextPlayer
 - NotifyPlayingAction
 - OnRaiseBet
 - PlayCard
 - PreviousBet
 - RaiseBet
 - RaiseRoundInformationChangedEvent
 - RaiseRoundOverEvent
 - RemovePlayedCards
 - Round
 - RoundInformationChangedListener
 - RoundOverEventListener
 - SetCurrentBet
 - SetCurrentTrucoCall
 - SetPlayers
 - SetViraCard
 - StartPlayingThread
 - StartRound
 - TakeCardFromTopDeck
- Nested Types

CG6TrucoView Class

- Fields
 - backgroundScoreBoard
 - backgroundTable
 - bmpBack
 - bmpDeck
 - buttonAcceptTruco
 - buttonNewGame
 - buttonPlayCard
 - buttonRejectTruco
 - buttonTruco
 - cardArea
 - cardClicked
 - cardH
 - cardsMap
 - cardsNameMap
 - cardW
 - controller
 - currentRound
 - hideCard
 - imageCardBack
 - imageDeck
 - memDCBack
 - memDCDeck
 - start
 - xOffset
 - yOffset
- Methods
 - ~CG6TrucoView
 - AssertValid
 - CG6TrucoView
 - CreateButton
 - DrawCards
 - DrawCurrentPlayerCards
 - DrawOtherPlayerCards
 - DrawScoreBoard
 - Dump
 - GetDocument
 - OnButtonAcceptTrucoClicked
 - OnButtonNewGameClicked
 - OnButtonPlayCardClicked
 - OnButtonRejectTrucoClicked
 - OnButtonTrucoClicked
 - OnDraw
 - OnInitialUpdate
 - OnLButtonDown
 - OnRButtonDown
 - OnRoundInformationsChangedEvent
 - PreCreateWindow
 - SetStatusBarText
 - ShowTrucoButton
 - TryPlayCard
 - UpdateButtons

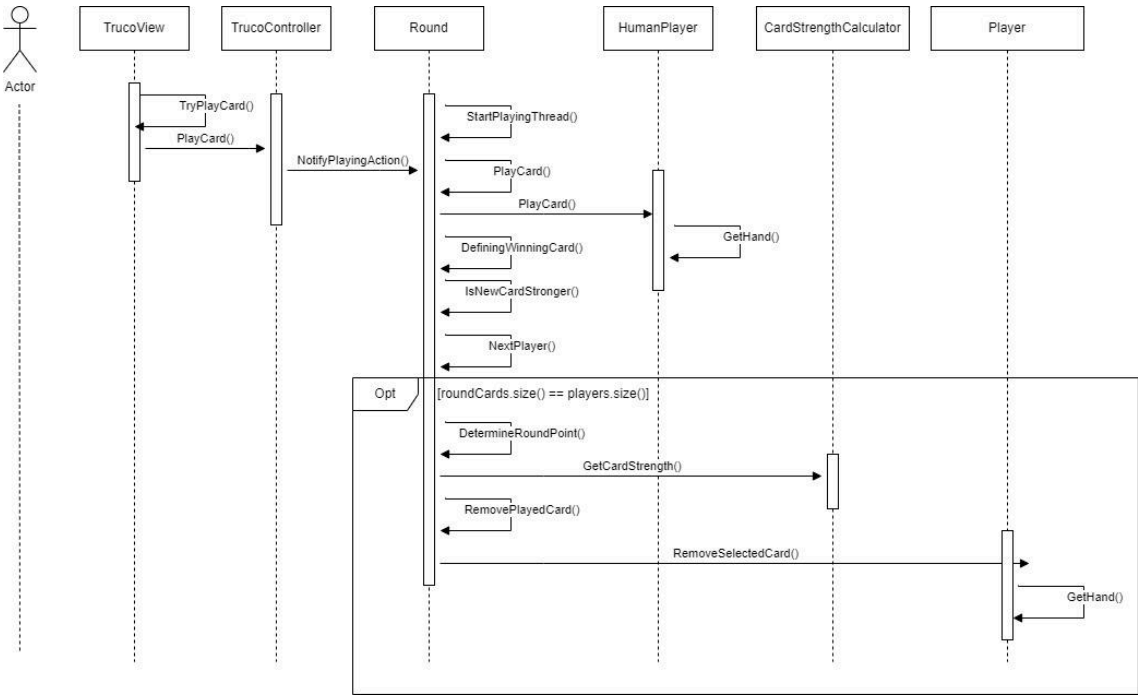
TrucoController Class

- Fields
 - activePlayerChangedEvent
 - firstPlayer
 - gamePoints
 - NUMBER_OF_PLAYERS
 - round
 - roundConditionVariable
 - roundMutex
- Methods
 - AcceptBet
 - ActivePlayerChangedEventListener
 - DenyBet
 - GetGamePoints
 - HandleRoundOver
 - IsActivePlayerHuman
 - PlayCard
 - RaiseActivePlayerChangedEvent
 - RaiseBet
 - RoundInformationChangedEventListener
 - StartGame
 - StartRoundHandlerThread
 - TrucoController
 - TrySetSelectedCardIndex

Diagrama de Sequência

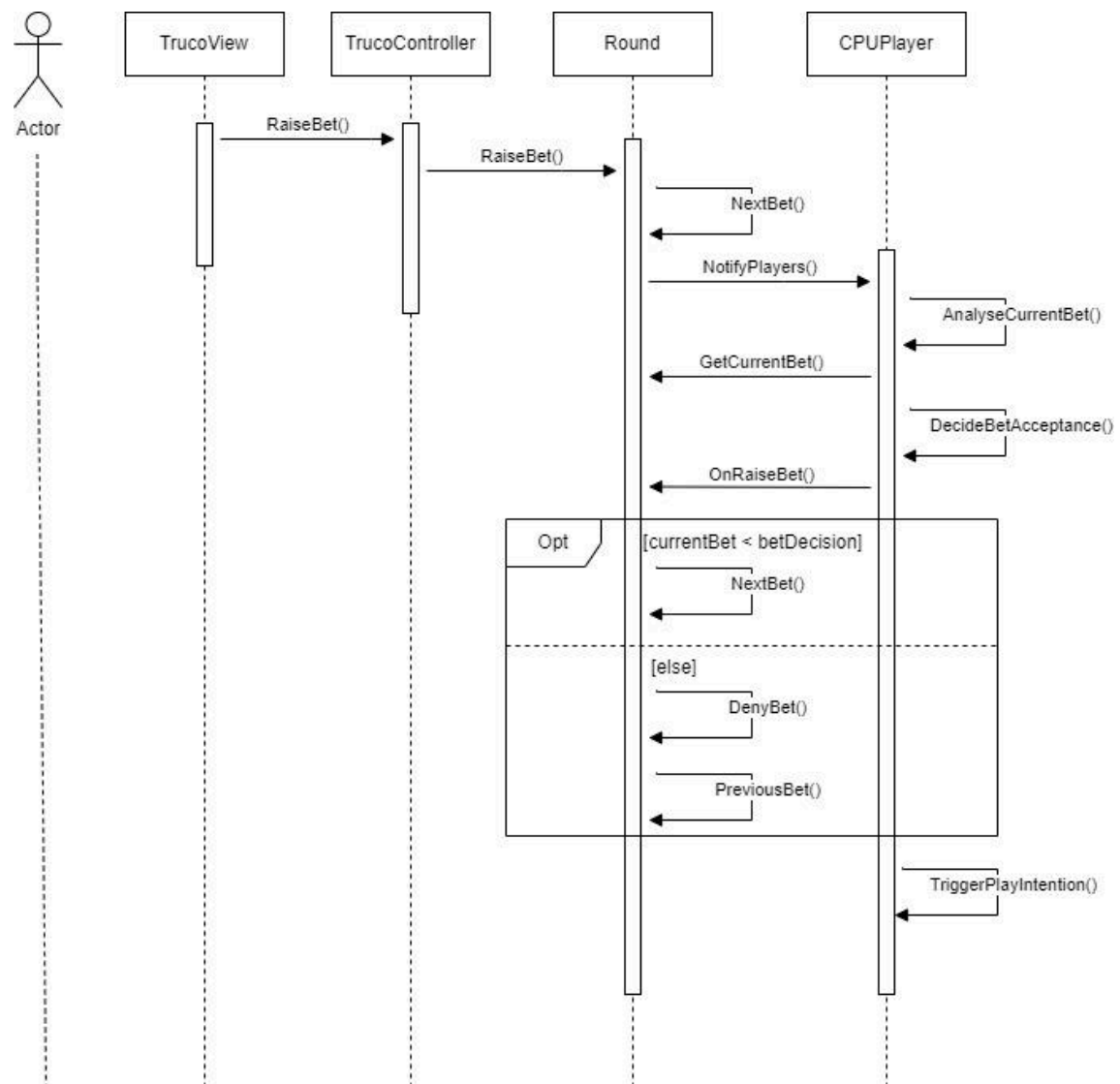
Dentre os fluxos de execução existentes no projeto, dois principais serão ilustrados nos diagramas de sequência a seguir.

O primeiro diagrama trata-se do fluxo completo de uma carta jogada por um humano.



Projeto – Truco

O segundo diagrama ilustra o fluxo percorrido quando uma ação de Truco é tomada pelo jogador humano.



Limitações e Decisões do Projeto

Para promover um desenvolvimento mais ágil e simplificar os cenários mapeados para o projeto, algumas limitações foram assumidas, conforme pode-se observar a seguir:

- Apenas o jogador humano irá efetuar a ação de Truco;
- O jogador CPU irá responder forma aleatória a ação de Truco, sem efetuar uma análise inteligente de jogo;
- Os times serão fixos: time humano contra time CPU;
- O time humano e CPU possuem jogadores com nomes fixos, não havendo possibilidade de troca/escolha dos mesmos;
- Após um time completar 12 pontos, o jogo voltará para a tela inicial;
- Na mão de 11 não haverá cartas descobertas, por conta de como a UI foi desenvolvida;
- Tanto os jogadores humanos quanto jogadores CPU não poderão jogar uma carta coberta;
- O ponto pelo empate de cartas será decidido pelo naipe;