FIT2102 – Assignment 2 Report

TwentyOne Game

**Name**: Nadeem Emadeldin Hamed Hamed Abdelkader

**Student ID**: 30146224

# Introduction

My TwentyOne AI implements a heuristic approach where it compares its hands against opponents up card to determine what’s the best action to take in order to enhance decision at each turn.

# Design of the code (including d struct) – How the AI decides which action to take

The player makes the decision by comparing the following

* The cards in its current hand
* The value of its current hands
  + using handCalc function
* The length of its current hand
  + Using length function
* The opponents up card rank
  + Using getRank function
* The opponents up card points value
  + Using toPoints function

Using the previous information, the player then follows the heuristic which was designed using long established twentyOne strategies to decide which of the 5 actions to take.

* Examples for each of the five actions
  + Hit
    - E.g., if the dealer’s up card is worth 10 points and the players current hand is worth 12-16, the player will hit
  + Stand
    - E.g., if the dealer’s up card is worth 10 points and the players current hand is worth 17 or more, the player will stand
  + DoubleDown
    - E.g. the player’s current hand is worth 11 exactly and the player has exactly 2 cards in hand, the player will DoubleDown
  + Split
    - E.g., if the player has a pair of aces and exactly 2 cards in hand, the player will split
  + Insurance
    - E.g., if it’s the first turn after the bidding turn and the dealers up card is an Ace, the player will take the Insurance action

# Memory and Parsing (including [BNF grammar](https://tgdwyer.github.io/parsercombinators/" \l "context-free-grammars-and-bnf))\*

# Functional Programming and Haskell Language Features Used (focus on why)

The following Haskell features were used when implementing my heuristic twentyOne player

* Pattern matching
  + Pattern matching was used to determine whether it’s the bidding round or not
    - By checking if the dealers up card is Nothing



* + - Here we can see that if the declares up card is Nothing then we will create a suitable build, this is made possible so elegantly due to Haskell’s pattern matching.
* Guards
  + Guards are used all over the program to create the heuristic because they act as a Boolean expression that if true will execute its body.
    - This allows for an elegant organization of if like conditions
* Maybe
  + Maybe is used in the program in 2 main places
    - The memory
      * The memory can either be a Nothing or Just String
        + This comes in handy because it isn’t always true that a memory exists and is in use.
    - The dealers up card
      * The dealer up card can either be a Nothing or Just Card
        + This comes in handy for example in the very first round (bidding round) because the dealer didn’t yet reveal his card, we can know that this is a bidding round. In other words, if dealers up card is Nothing, it’s the bidding round.
* Trace
  + This was vital as it was the main way of debugging and understanding the Skelton code
  + It was used to see how certain thing are stored like players info and current hand for example.

# References

* https://tgdwyer.github.io/

# How to Run\*

# Summary of the workings of the code\*

# High Level Overview of Design Decisions and Justification \*

# Future Work/Potential Improvements\*