Arondeep Khakh

F# BlackJack Project

# Project Choice and Description

For this project our group consisted of myself, Connor Rolfe, Conor Studley and Rafal Szponarski. The project we decided to create was the blackjack game.

For this project we were tasked with creating a SIMPLE blackjack game. We would use the F# Declarative language in order to implement the game. There are multiple different features required within the game one of them being the chance to play against other opponents as well as AI opponents.

# User identification and analysis.

When identifying our primary audience for the game there were a few things we took into consideration, one being the simplicity of the game. This would lead us to an older audience as often times a younger audience is interested in playing a more complex game, it only takes you being around older family members on a PC and watching them go straight to solitaire to understand this. This gave us an age range of 30-50 for the primary audience. The primary aim of the blackjack game would be to provide light entertainment for the audience, we have not implemented any form of betting within the game, so this removes any seriousness, fun only.

The secondary audience we established was the 18-30 group, we believe the project would entice this group less than the older group however we do believe that the blackjack game could be utilized by this age range. Potentially regular players could test different methods of playing the game in order to improve if they wished, so the game could also be utilized by the more serious player however, in a stress-free manner. We also believe the game could be utilized at social events potentially for light-hearted fun within this age range.

# Analysis of requirements and identification of features

What was required of us when creating this project was outlined within the task description. One of the key requirements was that the game was to be simple to use, negating any ideas of a complex implementation. Another one of the requirements for the game was to implement the ability to play against AI opponents that are able to play the game and make decisions for themselves.

The required simplicity of the system would also require a simple user interface, again our primary audience would be of people aged 30-50 so it is integral for us to make use of the game simple in order to allow easy access to such an age range who may not be quite as technically able as a younger audience who grew up around PC’S and games. In order to allow user input we decided to use a command line interface, the user will input simple text based commands in order to play the game for instance when it is a user’s turn they have the ability to fold, or to hold or to draw another card. This would be done by simply inputting ‘fold’ for example. The output of the system would also be handled via the text-based command line interface.

# User Story Specification

The two main features of the program are as follows, the selection of the amount of AI opponents the user wished to play against and the ability to take a turn which is when the user would specify whether they wish to ‘fold’, ‘draw’ or ‘hold’.

There are two potential paths the we identified as a group, the happy paths, these would be paths that do not produce an error and allow the user to continue playing the game. The second path would be the sad path, this is the path which would produce an error within the game and require the user to repeat a step they had taken incorrectly. An example of this would be inputting anything other then ‘draw’, ‘hold’ or ‘fold’ when taking a turn.

Using the Gherkin syntax, we have produced a user story specification in order to establish the users motivations whilst playing the game, this also allows us to identify the potential happy and sad paths within the game. You can see the Gherkin user stories below.

## Feature: Adding AI Opponents

As a user

I want to select the amount of AI opponents

In order to play the game against a different number each time

## Scenario: Adding opponents using a valid integer as an argument

If the user has started the game

And Is asked for input

When a valid integer is provided ‘x’

The game begins with ‘x’ amount of AI opponents

The user is dealt their first hand.

## Scenario: Adding opponents using an invalid integer as argument

If the user has started the game

And Is asked for input

When an invalid integer is provided ‘x’

Error message ‘E1’ is displayed

User asked for input again

## Scenario: Adding opponents using multiple valid integers as argument

If the user has started the game

And Is asked for input

When an invalid integer is provided ‘x’

Error message ‘E1’ is displayed

User asked for input again

## Scenario: Adding opponents a non-integer as argument

If the user has started the game

And Is asked for input

When an invalid integer is provided ‘x’

Error message ‘E1’ is displayed

User asked for input again

## Scenario: Adding opponents using no arguments at all

If the user has started the game

And Is asked for input

When an invalid integer is provided ‘x’

Error message ‘E1’ is displayed

User asked for input again

## Scenario: Adding opponents using multiple non-integers as argument

If the user has started the game

And Is asked for input

When an invalid integer is provided ‘x’

Error message ‘E1’ is displayed

User asked for input again

## Valid input examples

Input

* 0
* 1
* 2
* 3

# Feature: Taking a turn

As a user

I wish to take a turn

So that I can partake in the game

## Scenario: Using a single valid string as argument

Provided it Is the users turn

And the user is asked for input

When user inputs ‘decision’

The selected ‘decision’ will be executed

Results will be outputted to user

## Scenario: Using multiple valid strings as argument

Provided it is the users turn

And the user is asked for input

When the user inputs ‘decision, decision’

Error message will be displayed ‘E2’

User will be asked for input again

## Scenario: Using an invalid string as argument

Provided it is the users turn

And the user is asked for input

When the user inputs an invalid string

Error message will be displayed ‘E2’

User will be asked for input again

## Scenario: Using a singular non-string as argument

Provided it is the users turn

And the user is asked for input

When the user inputs a non-string as argument

Error message will be displayed ‘E2’

User will be asked for input again

## Scenario: Using multiple invalid non-strings as argument

Provided it is the users turn

And the user is asked for input

When the user inputs multiple invalid non-strings

Error message will be displayed ‘E2’

User will be asked for input again

## Scenario: Using no input as argument

Provided it is the users turn

And the user is asked for input

When the user inputs nothing

Error message will be displayed ‘E2’

User will be asked for input again

## Valid inputs

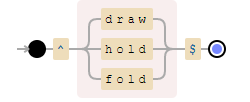
* Action
* Hold
* Draw
* Fold

|  |  |  |
| --- | --- | --- |
| ID | Error Message | Description |
| E1 | ERROR: Invalid input <input>  Expecting a single valid integer as input | This error message will be displayed when anything other than a single valid integer is inputted, anything other than 0,1,2,3. |
| E2 | ERROR: Invalid input  <decision>  Expecting a single valid command as input | This error message will be displayed when anything other than a single valid string command is inputted, anything other than ‘draw’, ‘hold’ or ‘fold’. |

# Data Model

## TurnInput

TurnInput is a representation of the action that the user inputs when they wish to take their turn. There are only 3 potential valid inputs here, ‘hold’, ‘draw’ or ‘fold’ anything other than those 3 strings would be an invalid input. Taking a turn relies on this data model in order to know which decision the user wishes to makes, for instance if the user were to ‘hold’ their turn would pass and the next user or AI would take their turn.



## Player

This data model is used to represent both the AI opponent and the user. A collection is used that contains an ID number (automatically assigned), the players hand and the player’s state. The user is automatically assigned an ID number of 0, any other opponents will be assigned 1,2 or 3 dependent upon how many players are playing. This data structure will be utilized within the Adding Ai Opponents feature, it will do so by creating the amount of players for the game that is requested by the user (NumOfOpponents). The taking a turn feature will also utilize this data structure when updating each players hand when they draw a card.



## NumOfOpponents

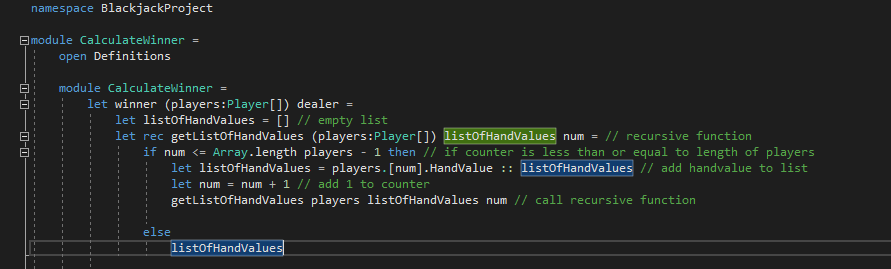
NumOfOpponents is used to represent the input the user provided when deciding the number of opponents. This data structure is represented using a positive integer value from 0 to 3. Anything other then those integers would be an invalid input, examples being ‘6’ or ‘none’.

# Implementation

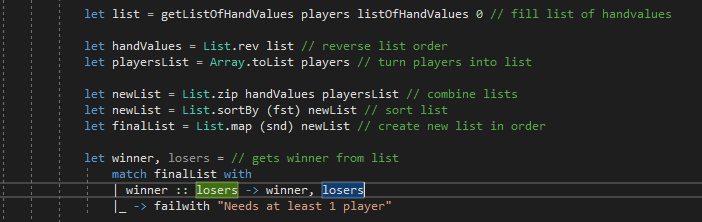
I utilized a few different functional programming techniques during the project, pattern matching being one of them which i used this within the calculate winner function in order to extract the winning player from the list of players.

I also used recursive functions within my code, these were used throughout my section of the code for various functions, an example being in the DealerTurn function I order for the dealer to keep drawing cards until a certain criteria is met. The reason why I used recursive function is because in functional programming it is better to do so rather than use iteration.

Finally, I utilized higher order functions in order to apply functions to other functions for example to compare to apply functions to all of the elements within a collection. I used list.SortBy in order to sort a list into a specific order, in this case determined by handvalue.

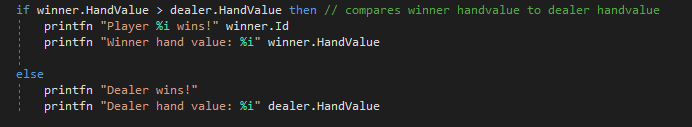


First of all, the winner function on line 7 takes the player array and the dealer and creates an empty list. Following this on line 9 we have a recursive function that takes an array of the players, listofHandValues and a counter. The recursive function utilizes an IF expression to compare the counter with the length of the players array, if it’s less then or equal to the players array it adds the handvalue of the player at the current position equal to the counter of the array, it is then added to the listofHandValues, it then increases the counter by 1 and calls itself. Otherwise the listofHandValues is returned.

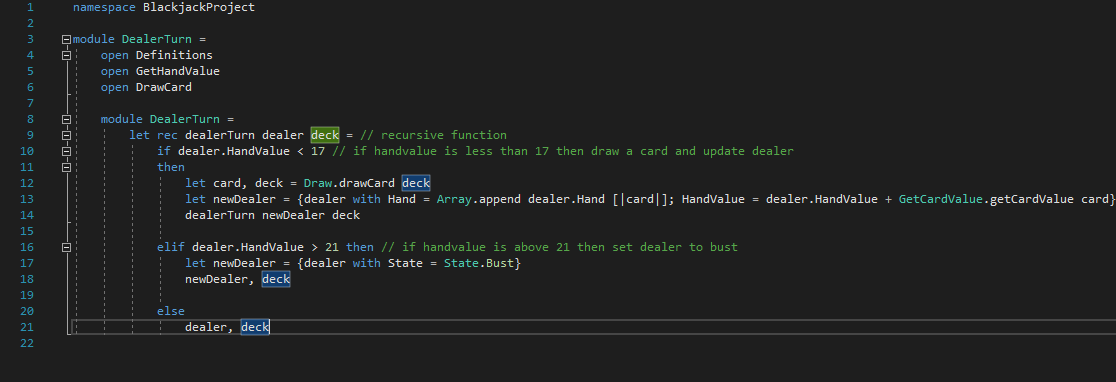


So on line 17 we are calling the above function and using it to fill the list. Line 20 flips the list we use the list.rev higher order funtion in order to do this. The array of players is then converted to a list using the Array.toList function from here we combine the list into a single list of tuples using the list.zip function it is then sorted based the first element of each tuple in the list using List.sortBy and finally we create a new list combining the second elements of the tuple from newList.

We then pattern match to pop the first element from the list as this is the player with the highest hand value (winner), failwith is an error message but with the current state of the code it will never be executed.



So here we are comparing the winners hand value to the dealers in order to determine the winner of the round, normally this code should return the winning player or dealer however due to time constraints I just returned a print statement.



The DealerTurn function is a recursive function that takes the dealer and the deck as arguments. The function then uses an IF expression to determine the hand value of the dealer, if the hand value is less than 17 it will draw a card and updates the dealer, it then calls itself , if it’s above 21 then the dealer’s state will be set to bust, it then returns an updated dealer and updated deck as a tuple.

TESTING

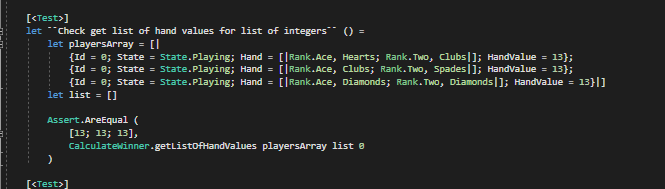
I referenced the different functions required to execute the tests, those being DealerTurn, Definitions and CalculateWinner. I had to create a new project, add a reference to BlackJackProject and had to install certain library packages, these packages were FsUnit, NUnit, NUnit3TestAdapter and Microsoft.NET.Test.Sdk.

Microsoft.NET.Test.Sdk allows me to create test project for .NET languages such as F#.

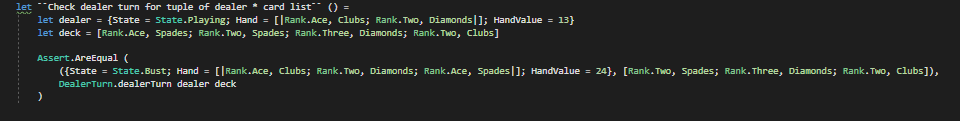
NUnit allows me to create unit test for .NET languages such as F#.

FsUnit is an extension of NUnit that allows unit tests to be more functional.

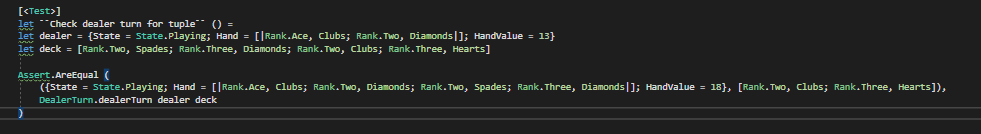
NUnit3TestAdapter allows NUnit tests to be executed within visual studio.



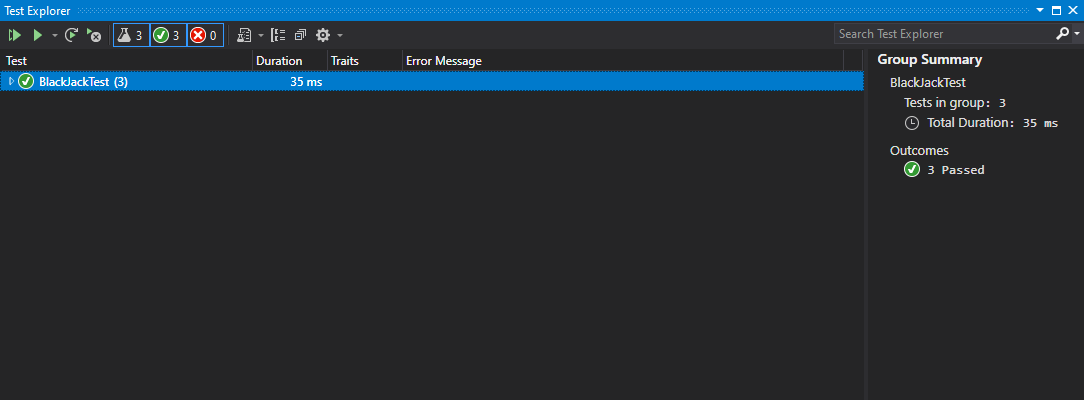
This test checks that a list of hand values is returned when the function is given an array of players and an empty list. The test succeeded and returned a list of 3 hand values when given an array of 3 players.



This test checks that the dealer turn function returns a tuple of card list and dealer when given a dealer and a deck. It also checks that the dealers state is set to bust because their hand value exceeded 21. The test succeeded and returned an updated dealer whose state was set to bust and an updated deck as a tuple.



This test checks the dealer turn function returns a tuple of dealer and card list, and also has a hand value between 17-21. The test succeeded and returns an updated dealer with a hand value between 17-21 and an updated deck as a tuple.



Above is a screenshot to confirm that my 3 tests worked.