CS246 Final Project Straights : demo.pdf

By Mark Liu

Fall 2021

I have implemented all the features of Straights and have different input files (.in) that can be used with my student number (**20897355**) as the seed to demonstrate that they work.

We first create the executable with:

```
make
```

This will create the executable called:

```
straights
```

Each of the following input files are used to demonstrate a specific feature and can be used with:

```
./straights  20897355 < name.in
```

For example:

```
./straights 20897355 < allcomputer.in
```

`allcomputer.in`

- This input file is used to demonstrate that the game works when every play is a computer.
- When prompted for human (h) or computer (c), it will input c for all players.
- Running:

  ```
  ./straights 20897355 < allcomputer.in
  ```

  we see Player 3 will win with a score of 14.

`allhuman.in`

- This input file is used to demonstrate that the game works when every play is a human.
- When prompted for human (h) or computer (c), it will input h for all players
- Valid moves are inputted for one full round. Then each player will ragequit and be replaced by computer players.
- Running:

  ```
  ./straights 20897355 < allhuman.in
  ```

  we see Player 1 will win with a score of 38.

`onehuman.in`

- This input file is used to demonstrate that the game works when there is one human player and three computer players.
- When prompted for human (h) or computer (c), it will input h for player 1 and c for players 2, 3, 4.
- The human player 1 will input valid moves until the game is finished.
- Running:

```
./straights 20897355 < onehuman.in
```

  we see Player 2 will win with a score of 15.

`deck.in`

- This input file is used to demonstrate that the game works when the ‘deck’ command is used.
- It will input 2 human players and 2 computer players.
- The first human player will input the ‘deck’ command 2 times in a row and then ragequit.
- The remaining human player will input the deck command 1 time and then ragequit.
- Running:

```
./straights 20897355 < deck.in
```

  we see Player 3 will win with a score of 14.

`quit.in`

- This input file is used to demonstrate that the game works when the ‘quit’ command is used.
- It will input 1 human player and 3 computer players.
- The human player will play several valid moves and then quit. The game will stop immediately with no winners.
- Running:

```
./straights 20897355 < quit.in
```

  we see the game will stop immediately with no winners.

`ragequit.in`

- This input file is used to demonstrate that the game works when the ‘quit’ command is used.
- It will input 3 computer players and 1 input player.
- The human player will play several valid moves and then ragequit. A computer player will take over.
- Running:

```
./straights 20897355 < ragequit.in
```

we see Player 3 will win with a score of 60.

`allragequit.in`

- This input file is used to demonstrate that the game works all the player's ragequit.
- It will input 4 human players.
- All human players immediately ragequit. Each player is take over by a computer player.
- Running:

  ```
  ./straights 20897355 < allragequit.in
  ```

  we see Player 2 will win with a score of 26.

`invalids1.in`

- This input file is used to demonstrate that the game can handle the user playing cards that they don't have.
- It will input 3 computer players and 1 human player.
- When it is the human player's turn, they will try to play:
  - Cards that don't exist (such as 'ZZ' and '19')
  - Cards that they don't have in their hand (such as 'TD' and '1S')
  - Cards that they have in their hand but are not legal (such as '1H' and '5C')
- Output will be printed to tell the user that these are not valid. It will continue to prompt the user to play a valid card.
- Running:

  ```
  ./straights 20897355 < invalids1.in
  ```

  we see Player 3 will win with a score of 14.

`invalids2.in`

- This input file is used to demonstrate that the game can handle the user trying to discard cards that they don't have.
- It will input 3 computer players and 1 human player.
- When it is the human player's turn, they will try to discard several cards that are not in their hand (such as '7H' and 'TD') and cards that don't exist (such as 'QQ' and '00').
- Output will be printed to tell the user that these are not valid. It will continue to prompt the user to discard a valid card.
- Running:

  ```
  ./straights 20897355 < invalids2.in
  ```

  we see Player 3 will win with a score of 61.

`invalids3.in`

- This input file is used to demonstrate that the game can handle the user trying to play cards when they have no legal moves.
- It will input 1 human player and 3 computer players.
- When it is the human player's turn, it will try to play cards that are not legal.
- Output will be printed to tell the user that these are not valid. It will continue to prompt the user until they discard a card in their hand.
- Running:
  ```
  ./straights 20897355 < invalids3.in
  ```
  we see Player 2 will win with a score of 59.

`invalids4.in`

- This input file is used to demonstrate that the game can handle the user trying to discard cards when they have legal moves.
- It will input 3 computer players and 1 human player.
- When it is the human player's turn, it will try to discard several cards.
- Output will be printed to tell the user that they have a legal play that they must play. It will continue to prompt the user until they play a legal card.
- Running:
  ```
  ./straights 20897355 < invalids4.in
  ```
  we see Player 3 will win with a score of 61.

`invalids5.in`

- This input file is used to demonstrate that the game can handle the user trying to enter invalid commands.
- When prompting for human (h) or computer (c), it will input invalid commands such as 'z' and '2'. Output will be printed to tell the user they must enter h or c.
- When it is the human player's turn to play or discard a card, it will input invalid commands such as "asdf", "123123", "PLAY 7C", "playy 7c", "rage quit". Output will be printed to tell the user these are invalid.
- Running:
  ```
  ./straights 20897355 < invalids5.in
  ```
  we see Player 3 will win with a score of 14.

I have also implemented two enhancements to the game. The first is called 'loadMode' which allows the game to start execution from a saved state. The second is called 'changeStrat' which allows the user to decide the strategy that the Computer Player(s) should use when making their moves.

Both of these keywords can be passed as command line arguments (not at the same time) after the seed. Passing a seed is required for these enhancements.

Command to run a game starting from a saved game state:

```
./straights [SEED] loadMode < [STATETOLOAD].in
```

Note:

- The files that contain a game state to load, such as LOADEDSTATE1.in, LOADEDSTATE2.in, LOADEDSTATE3.in are formatted differently than the input files mentioned above such as allcomputer.in, quit.in, invalids1.in, etc. Trying to use them interchangeably, such as
  ```
  ./straights 20897355 loadMode < invalids1.in
  ```
  will not work.

For example, we can run a game starting with the game state saved in LOADEDSTATE1.in using:

```
./straights 20897355 loadMode < LOADEDSTATE1.in
```

The game state in [STATETOLOAD].in files must follow this format:

```
Cards Currently Played on the Table
Player 1’s Total Score (excluding score from current discards)
Player 2’s Total Score (excluding score from current discards)
Player 3’s Total Score (excluding score from current discards)
Player 4’s Total Score (excluding score from current discards)
Player 1’s Cards in Hand in Current Round
Player 2’s Cards in Hand in Current Round
Player 3’s Cards in Hand in Current Round
Player 4’s Cards in Hand in Current Round
Player 1’s Discarded Cards in Current Round
Player 2’s Discarded Cards in Current Round
Player 3’s Discarded Cards in Current Round
Player 4’s Discarded Cards in Current Round
ID of Current Player’s Turn
Player Moves Starting with Current Player’s Move
```

Note:

- Newlines are used to separate each of these pieces of information

- None of these pieces of information can be omitted. That is, there must >= 1 card currently played on the table, there must be >= 1 card in each player's hand, and there must be >= 1 card in each player's discarded cards pile.
- Player 1 has Player ID of 1, Player 2 has Player ID of 2, etc.

For an example, see LOADEDSTATE1.in:

```
7H 8H 9H 6H 7S 8S 7D 8D 9D TD
62
69
70
78
1C 2D
4D 1S
1H JD
QD 4C
1D
2D 3S
2H
5D 5H
1
discard 1C
discard 1S
play JD
play QD
discard 2D
discard 4D
discard 1H
discard 4C
```

Explanation of LOADEDSTATE1.in:
This simulates a game with the following cards played on the table:

```
Clubs:
Diamonds: 7 8 9 T
Hearts: 7 8 9 6
Spades: 7 8
```

And with the following Player states:
Player 1:

```
Score: 62
Cards In Hand: 1C 2D
Currently Discarded Cards: 1D
```

Player 2:

```
Score: 69
Cards In Hand: 4D 1S
Currently Discarded Cards: 2D 3S
```

Player 3:

```
Score: 70
Cards In Hand: 1H JD
Currently Discarded Cards: 2H
```

Player 4:

```
Score: 78
Hand of Cards: QD 4C
Currently Discarded Cards: 5D 5H
```

Following this, we read in `1`, which tells us that it is Player 1's turn.
Thus, the following commands tell us:

```
Player 1 discards 1C
Player 2 discards 1S
Player 3 plays JD
Player 4 plays QD
Player 1 discards 2D
Player 2 discards 4D
Player 3 discards 1H
```

Using `./straights 20897355 loadMode < LOADEDSTATE1.in`, we see Player 1 wins with a score of 62+4=66.

We can use this ability to load game states to test the case where multiple winners.
For example:

`LOADEDSTATE2.in`

- This input file is used to demonstrate that the game can handle 2 winners.
- Running:
- `./straights 20897355 loadMode < LOADEDSTATE2.in`
  we see that Player 1 and Player 3 both win with a score of 25.

`LOADEDSTATE3.in`

- This input file is used to demonstrate that the game can handle 3 winners.

- Running:

```
./straights 20897355 loadMode < LOADEDSTATE3.in
```

  we see that Player 1, Player 2, and Player 3 all win with a score of 48.

I have also implemented a feature to change the Strategy that the Computer Player uses to make move during their turn.
Command to use changeStrat feature manually:

```
./straights [SEED] changeStrat
```

Command to use changeStrat with an input file that contains the new strategy(see changeStrat1.in):

```
./straights [SEED] changeStrat < [INPUTFILE].in
```

This will run the game regularly (not in loadMode) and will also prompt the user for the strategy they want each Computer Player to use. The user will then be able to input 1 or 2:

  1 = Normal Strategy (play first legal play in hand, if there are no legal plays, discard first card in hand)
  2 = Drop Lowest Strategy (play first legal play in hand, if there are no legal plays, discard card in hand with the lowest rank)

We can see this in changeStrat1.in:

```
c
1
c
1
c
2
c
2
```

This creates 4 computer players. Players 1, 2 use the original 'Normal' Strategy and Players 3, 4 use the 'Drop Lowest' Strategy.
Running

```
./straights 29897355 changeStrat < changeStrat1.in
```

we can see the following sequence in the last round:

```
================================PLAYER1'S TURN:================================
Cards on the table:
Clubs:
```

```
Diamonds: 7 8 9 T 6 5 J
Hearts: 7 8 9 6 T J
Spades: 7 8 6 5 4 3 2
Your hand:
 JC
 2D
 TS
 3D
 2H
 KS
Legal plays:
Player1 discards JC
================================PLAYER2'S TURN:================================
Cards on the table:
Clubs:
Diamonds: 7 8 9 T 6 5 J
Hearts: 7 8 9 6 T J
Spades: 7 8 6 5 4 3 2
Your hand:
 1D
 4C
 6C
 QS
 QH
 1C
Legal plays:
 QH
Player2 plays QH
================================PLAYER3'S TURN:================================
Cards on the table:
Clubs:
Diamonds: 7 8 9 T 6 5 J
Hearts: 7 8 9 6 T J Q
Spades: 7 8 6 5 4 3 2
Your hand:
 4H
 KD
 3C
 KC
 5C
 8C
Legal plays:
Player3 discards 3C
```
As we can see, Player 1 uses the ‘Normal’ Strategy by discarding a Jack instead of a 2. Player 3 uses the ‘Drop Lowest’ Strategy by discarding a 3 of Clubs instead of a 4 of Hearts which was the first card in their hand.