CS51 Project: Draft Specification

Project name: Japanese Mahjong Hand Helper
Name: Mason Tan
Email address: mtan810@gmail.com

Brief Overview

Japanese Mahjong, also known as Riichi Mahjong, is a popular variation of mahjong played in… well, Japan. Those who played any variation of mahjong (besides mahjong solitaire, no offense) may agree with me that mahjong is a pretty complex game filled with many rules and strategies. Because of the nature of the game, someone who wants learn and get into it may have a difficult time in the beginning. So I thought it would be interesting to develop a learning application for beginners as well as existing players.

The idea behind this application is to guide players in creating a winning mahjong hand. Players should at least know the tiles in the game (please refer to http://en.wikipedia.org/wiki/Japanese_Mahjong for more info). A winning hand consists of 4 sets and a pair (same 2 tiles). A set consists of either a triple (same 3 tiles) or a sequence of 3 tiles of the same suit, such as 123, 567, etc. There is a special rule for quadruples (same 4 tiles) where a player can form one and draw an extra tile from the wall, but the tiles in the quadruple cannot be used to make other sets in the hand. We can ignore this for the time being. In this application, the player will start off with 14 tiles. If the hand is already a winning hand, then congrats (chances are VERY low however)! If not, then the player discards a tile and the turn ends. Then in the beginning of each turn and every turn, the player will draw a tile from the "wall," which contains all the leftover tiles, and the same process from before repeats itself. The game goes on until the player wins with a complete hand or lose due to the lack of tiles left to draw.

The main goal of this application is to help players, whether they are beginners or not, develop tile efficiency. By practicing with this application, players should gain more experience in determining which tiles to keep and which tiles to throw out in order to maximize the chances of winning.

Feature List

Core features:

- Backtracking algorithm (http://en.wikipedia.org/wiki/Backtracking): The main algorithm of this application. The purpose of this algorithm is to determine whether the player's current hand is a winning hand or not. The algorithm will scan all the combinations of sets and pairs the hand can make. If it is able to find 4 sets and a pair, then the hand is a winning hand.
- Hand sort algorithm: The purpose of this algorithm is to sort the player's hand by suit and numbers. It will also consider the wind tiles and dragon tiles. Different types of tiles will be explained later in the Technical Specification section.
- Draw & discard algorithms: The purpose of these algorithms is to add and remove a tile from the player’s current hand.
- UI: The terminal will display the current hand and the tile drawn in the beginning of each turn in text form. The player will type in the terminal the tile to discard. The terminal will then display the new sorted hand (or the player’s old hand if the player decides to discard the drawn tile instead) and the new drawn tile. This process repeats itself until the game is over.

Cool extensions:

- GUI: Improvement to the UI. Rather than using text to depict the tiles, using actual pictures would be better visually. And rather than having to type out which tile to discard, it would be better if the player can just click the tile instead.
- Website: Make the project host-able on a website.

Technical Specification

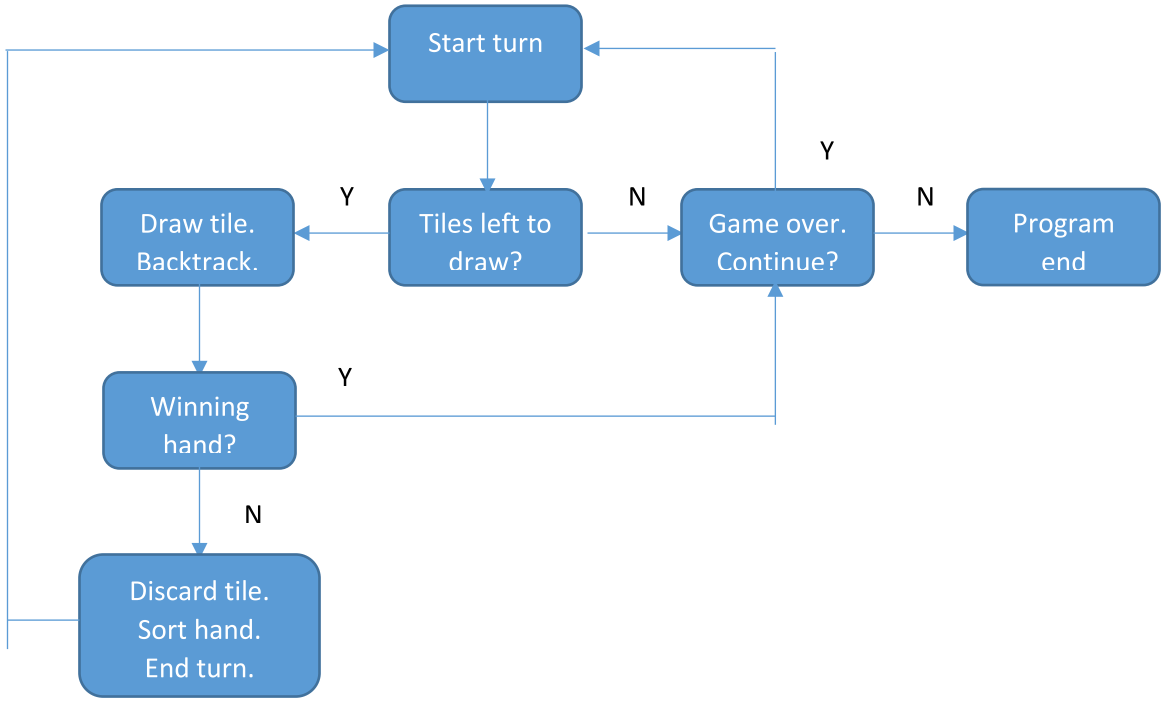
To start off, a module for tiles is required since the game is based mainly on them. There are many different kinds of tiles in Japanese mahjong, up to 136 tiles in total and 34 different kinds of tiles, with four of each kind. There are three suits: pin (circles), sou (bamboo), and man (characters). Each suit has a number ranging from one to nine. The name for tiles with a suit and number follow the pattern "number" + "suit" (for example, 4pin/4circles/4p). There are also honor tiles which are divided between wind tiles (East, South, West, and North) and dragon tiles (White, Green, and Red). Honor tiles do not have a suit and number. The tile module will include all these different types of tiles.

```
module type tile =
  sig
    type suit
    type value
    type honor
  type tile
end
```

The next module, which is the main module of the project, is the player's hand. This is where all the algorithms from the core features will reside. Since a hand consists of multiple tiles, it will use the tile module explained earlier. The tile module could be combined with the hand module, but is separated to make things clearer.

```
module type hand =
  sig
    type hand
    val sort : hand -> hand
    val draw : tile -> hand -> hand
    val discard: tile -> hand -> hand
    val backtrack: hand -> bool
  end
```

First, the main program will initialize and randomize all 136 tiles. Then it will create a sorted hand by drawing 14 tiles from the wall. It will also call 'backtrack' to determine if the hand is a winning hand or not. If it is, then the player wins and the program will either terminate or restart, depending on the player’s input. If not, then the program will prompt the user to discard a tile. After discarding a tile, the program will call 'sort' to sort the current hand and 'draw' to draw the next tile. The process repeats itself until the game is over. Here is a simple flow diagram of an average turn:



The hand sort algorithm will sort the hand in a way to make it easier for the players to read. The order of the tiles will be from left to right: 1-9 characters, 1-9 circles, 1-9 bamboos, East, South, West, North, White, Green, and Red.

The draw algorithm will take a tile from the wall and to the player's current hand. If there are no more tiles in the wall to draw, then the player loses and the game is over. The discard algorithm will discard a tile from the player's current hand. It will keep track of all the discarded tiles so the player can see which tiles were discarded.

The Backtracking algorithm will find all solutions to the winning hand. The idea behind the Backtracking algorithm is that it incrementally builds candidates to the solutions. It abandons each partial candidate as soon as it determines that the candidate cannot possibly be completed to a valid solution. In the case of mahjong, this algorithm will try to find and keep track of either the first set or pair. If it finds a pair first, then it will continue to look for the remaining sets. If it finds a set first, then it will continue to look for the remaining sets and a pair. It continues until all four sets and one pair are found, and backtracks if they are not found. Of course, it may be impossible to find a valid solution, meaning that the hand is not a winning hand.

Perhaps one issue with the algorithm may be its run time. Because it scans for all combinations of sets and pairs that can be formed with a hand, it might take some time for it to finish.

Next Steps

This project will be written entirely in Ocaml. The GUI described in the Cool Extensions of the Feature List section would be written in PHP, HTML, and Javascript if the project reaches that stage.