CS51 Project: Report

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

Brief Overview

Japanese Mahjong Hand Helper is a learning application that allows players to practice forming a winning hand. A winning hand consists of a pair and four sets; a set is either a triple or a sequence.

The application starts with a hand which consists of thirteen tiles. During each turn, a player draws a tile and the application checks to see if a hand is a winning hand. If it is, then congratulations to the player! If not, then the player chooses which tile to discard and the turn ends. This process repeats until the hand is a winning hand or there are no more tiles left to draw.

To check the demo video, please visit this link:
https://youtu.be/r2_-7uJg_aA
Please note that the tiles are rigged for demo purposes.

I had set up a git repository on code.seas for this project and granted the CS51 staff read access. Here is the link to the project:
https://code.seas.harvard.edu/masontans-cs51-project
And here is the link to the repo:
https://code.seas.harvard.edu/masontans-cs51-project/masontans-cs51-project

Original Planning

The original plan includes two different modules, the tile module and the hand module. The tile module is used to define the two different types of tiles: num+suit tiles and honor tiles. The hand module is used to define the hand and the four main functions of the application: sort, draw, discard, and backtrack.

Milestones

Draft Specification: At this point I set myself goals for the project and expanded my ideas with more details. Please refer to the Draft Specification pdf for more details.

Technical Specification: At this point I have defined 'tile' to be a tuple (num*suit), but I have yet to figure out how to define 'tile' if it was an honor tile. By the end of this milestone, I realized that I need to define 'tile' more clearly.

Checkpoint: At this point I changed up my design and interfaces multiple times. Here I combined both the tile and hand modules into one single hand module. I also combined the num, suit, and honor types into 'tile'. 'tile', which was a tuple, is now a record with type num, suit, honor, and flag. With a better defined 'tile' type, I was able to make better progress. Redefining 'tile' properly here is the best choice I made in this project.

Final: To my surprise, I was able to complete the main four functions and their tests mainly because of a much more defined 'tile' type. I was able to complete the game with a minimal UI. At this point, I realized there is not enough time for the cool extensions specified in the Draft Specification. Instead, I used the ocaml graphics library to draw circles with different colors based on the tile type.

Specs

Here is the original draft of the tile and hand modules:

Tile module	Hand module
1. module type tile = 2. sig 3. type suit 4. type value 5. type honor 6. type tile 7. end	1. module type hand = 2. sig 3. type hand 4. val sort : hand -> hand 5. val draw : tile -> hand -> hand 6. val discard: tile -> hand -> hand 7. val backtrack: hand -> bool 8. end

Here are the final specs, with the tile and hand modules combined:

Hand module
1. module type HAND = 2. sig 3. type num

```
4.  type suit
5.  type honor
6.  type flag
7.  type tile
8.  type hand
9.  val string_of_num : num -> string
10. val int_of_num : num -> int
11. val string_of_suit : suit -> string
12. val string_of_honor : honor -> string
13. val print_tile : tile -> unit
14. val print_tiles : tile array -> unit
15. val generate_wall : tile array
16. val generate_wall_test : tile array
17. val shuffle : tile array -> unit
18. val generate_hand : tile array -> tile array
19. val sort : tile array -> unit
20. val draw : int -> tile array -> tile array -> unit
21. val discard : tile array -> unit
22. val track_pair : tile array -> int -> unit
23. val track_set : tile array -> unit
24. val verify : tile array -> bool
25. val unmark : tile array -> unit
26. val backtrack : tile array -> int ref -> bool
27. val draw_tile : tile -> int * int -> unit
28. val draw_hand : tile array -> unit
29. val print_test : tile array -> unit
30. val run_tests : unit
31. end
```

Also included in the final specs is the game interface:

Main

```
1. val game : int -> int ref -> tile array -> tile array -> unit
2. val game_start : string -> unit
3. val start_ui : unit
```

In the final specs, the interface still includes all the specs from the original draft, but has evolved into a larger interface that includes many functions that were not part of the original draft:

- Conversion functions (string_of_num, int_of_num, string_of_suit, string_of_honor) are used to convert the num/suit/honor types to int/string which are then used in many parts of the module.
- Print functions (print_tile, print_tiles, print_test, run_tests) are used to print the tiles and test the module.
- Generate functions (generate_wall, generate_wall_test, and generate_hand) are used to generate the wall and hand in the game.
- Shuffle function (shuffle) is used to shuffle the wall. This is implemented using the Fisher-Yates shuffle (for more details, visit http://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle).
- Backtrack helper functions (track_pair, track_set, verify, and unmark) are all functions that are integral to the main backtrack function.
- Graphic functions (draw_tile, draw_hand) use the ocaml graphics library to draw circles with different colors based on tile types.
- Main functions (game, game_start, start_ui) are used to start the game itself.

Please refer to hand.ml and main.ml for the actual code implementation. Please refer to the Technical Specification & Checkpoint pdfs for more specific details on all the updates and changes throughout the project.

Lesson

During the Checkpoint milestone, I learned that designing the interfaces properly in the beginning is important. This is the most important thing I learned from the project. Even though this should have been obvious, I did not realize it until I was in the middle of the project. In the future, I will pay more attention to the design and interfaces before heading straight to the implementation. Not only will this make life much easier but it also will save a lot of time.

To future CS51 students and to anyone in general, please take my lesson into consideration for your future endeavors!

Future

Even though the core features of the project are finished, there are many other features and improvements I wanted to make if there were more time. More specifically, I wanted to complete the cool extensions of the project.

I would like to improve the UI. For example, rather than having to type which tile to discard each time, it would be better if players can click on the tiles instead. The project can also be improved aesthetically. This includes adding actual pictures of the tiles rather than using plain text.

Also, I would like to host this application on a website where players can play online rather than having to install, compile, and run the application. In the end, not everyone runs a Linux distro and has ocaml installed.

Lastly, I would like this to be part of a bigger application where players can actually play Japanese Mahjong with each other.

These ideas are great for future projects and something I would definitely recommend looking into in the future!