COMP 3023 – Assignment 1

Sushi Slam System Design

Game Class

The Game class drives the game and after instantiation, creates the shuffled deck and both players. It loops three times, each time creating two hands of cards and subsequently calling round(), which calls turn() ten times, after which it performs end of round scoring, hand clearing and tableau clearing. Turn() manages presentation of hands and adding to the tableaus. The Game class has five private member variables: _hand1, _hand2, _player1, _player2 and _roundNumber. Each player is always working with the same hand, but each _hand vector is copied at the end of a turn (each with one less card than the start of the turn) and the copied hand is reassigned to the player. This was a clean and readable way to manage this nuance of the game. Based on the format of the game, it was intuitive to create the two ‘major’ functions that drive the game: turn() and round() in a hierarchical manner. Game has a 1 to 1 or many relationship with Player class and a 1 to 1 relationship with the Deck class. Calling the destructor on Game will also destroy Player and Deck. Game needs to include the Deck class.

Deck Class

The Deck class has a private member variable called _cards which is a collection of cards. It includes member functions for populating and shuffling the _cards collection. Its destructor deletes all the cards in its collection. The Deck class holds a type definition for the CardCollection so itself and all classes that include it have access to the type definition. Deck needs to include the Card class and all its sub-classes.

Player Class

The Player class has four private member variables: an array of names called _names, _name, _playerScore and _tableau. _name is initialized when a Player is constructed based on a random choice from the array. It includes member functions to add a card to its tableau and to clear its own tableau. It calculates scoring for the round by looping through each card in a reference to its tableau and the other players tableau and checking the card type. By usingreferences, we avoid copying the tableau which can provide a performance benefit. If the card type has yet to be scored, then it scores all cards of its type in the tableau and marks the card type as ‘scored’ in another vector. A special condition has been added in the case where no Maki rolls are present, so three points can be awarded. The Player class holds a type definition for the CardCollection so itself and all classes that include it have access to the type definition. It needs to include the Card class.

Card Class

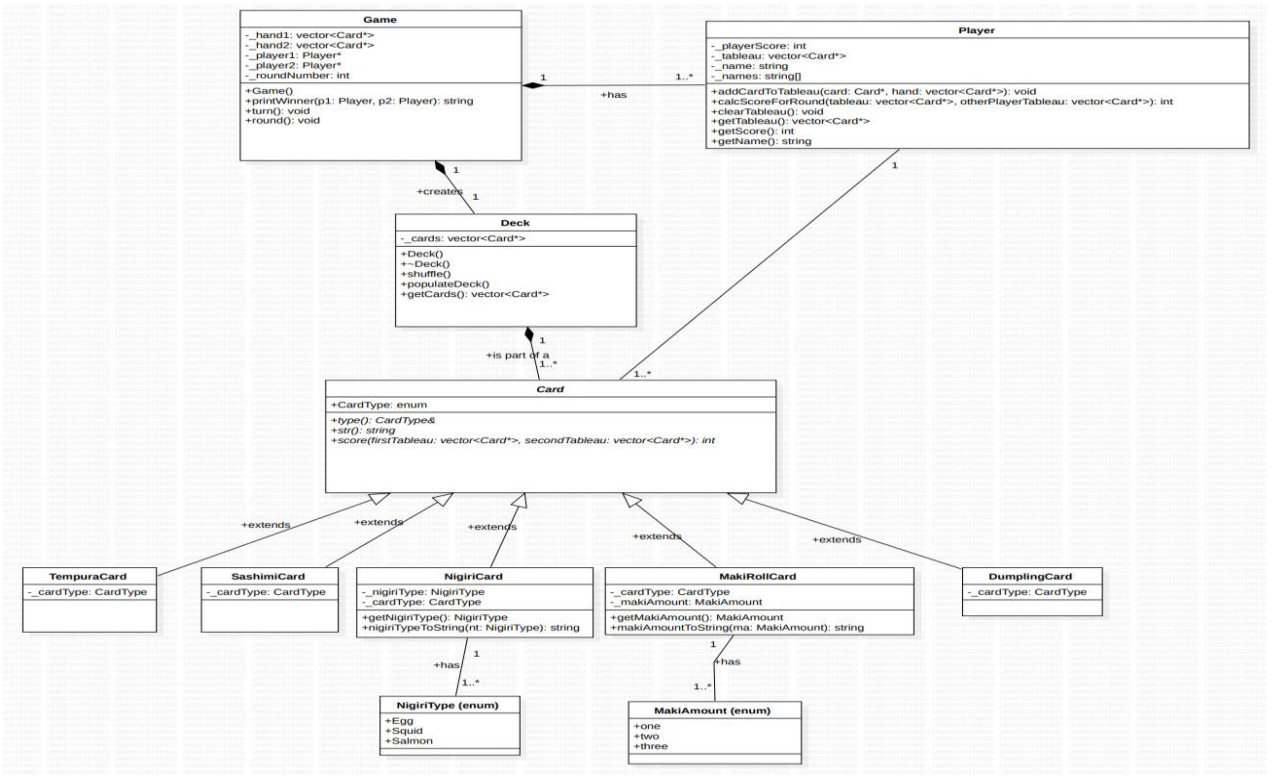
The Card class is abstract and cannot be instantiated. It has a public enum called CardType, its values being the types of cards of its sub-types. As it cannot ever be instantiated (and only sub-types are ever instaniated), the 'publicness' of CardType is not a significant encapsulation issue. It contains three virtual member functions overridden by the child card classes.

Card sub-types (DumplingCard, MakiRollCard, NigiriCard, SashimiCard, TempuraCard)

Each sub type predictably has functions that overide the parent abstract class. One returns the enum CardType, one returns a string representation of this CardType and one (called score) calculates the score of all its type in the tableau of the current round. Each score implementation is quite different due to the diverse ways each type of card rewards points to the player. MakiRollCard sub-type has an additional member variable that defines the number of maki rolls on the card. Enum classes are used in both cases. NigiriCard sub-type also has an additional member variable that defines the type of Nigiri on the card. Each card sub-type includes the parent Card class and the Player class

Class Diagram

***There is a larger (A5) PDF version of the class diagram saved in the repository.**



*** My original design included four additional classes that I deemed not necessary (Tableau, Score, Hand & Round). As I worked through implementing the system I felt including them would increase complexity. Removing them has greatly simplified my system and class diagram as a result.**

***My first few GIT commits may have my previous TAFE email in the logs. I usually develop on my MAC and have not performed any GIT actions on my Windows machine for some time. I modified the email over to my current UniSA one after I noticed this.**