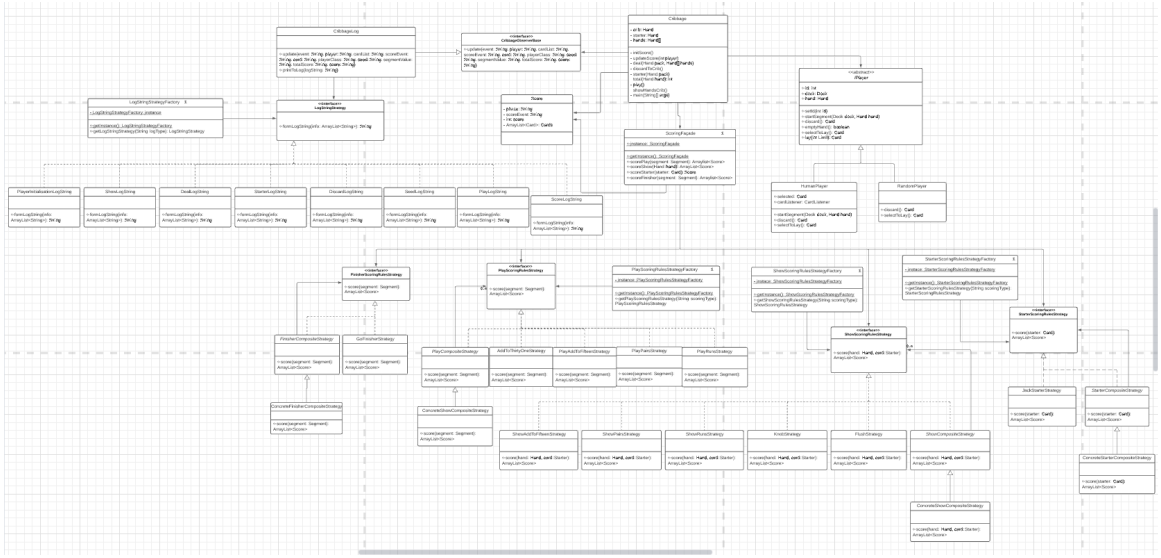
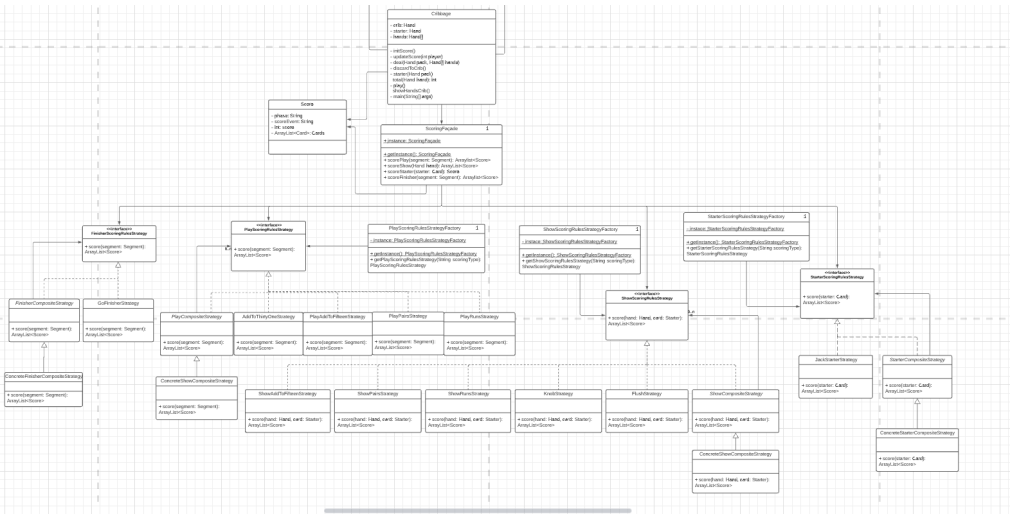
Project **2 Report** - Aidan Western, Yunhao Fang, Mahee Hossain

Full Design Diagram



Scoring System



The implementation of the scoring logic for the Cribbage Trainer was done by creating a scoring subsystem, contained within a Scoring package separate from the provided Cribbage package. As such, the scoring system required a Facade pattern, implemented with the class ScoringFacade, in order to maintain low coupling with the base Cribbage class, as it simply has to call the scoring function appropriate for the current phase of the round; either the play, show or starter (dealing cards, discarding cards, drawing starter card, etc.) phases. This enforces protected variation by allowing for the alteration of the scoring subsystem without requiring much, if any, alterations to the base Cribbage class.
The creation of this Facade pattern was also done to maintain high cohesion in the originally provided classes, leaving the basic operation of the Cribbage round to them whilst outsourcing the scoring logic to the scoring package we implemented.

The scoring subsystem itself primarily consists of four separate strategy pattern implementations; which are accessed through the StarterScoringRulesStrategy, PlayScoringRulesStrategy, FinisherScoringRulesStrategy and ShowScoringRulesStrategy interfaces, which respectively contain scoring rules for the start of a round, the play phase, finisher (specifically the scoring for playing the last card in a segment without reaching 31) and the show phase.

Both the StarterScoringRulesStrategy and FinisherScoringRulesStrategy patterns only contain one strategy each: the JackStarterStrategy and GoFinisherStrategy respectively, as well as a composite strategy each. Originally both of these strategies were contained within the PlayScoringRulesStrategy pattern, but the decision was made to place them into their own strategy patterns, in order to allow for the potential inclusion of new scoring strategies that similarly take place during the start of the round or end of the play phase, respectively. This was also the reason for the inclusion of a concrete composite class for the two patterns, to allow for the handling of multiple round start scoring rules happening at once if more are implemented in the future.
Similarly, the strategy pattern was employed for the play phase scoring rules and show phase scoring rules to allow for the potential introduction of new scoring rules without requiring alterations to the base Cribbage class.

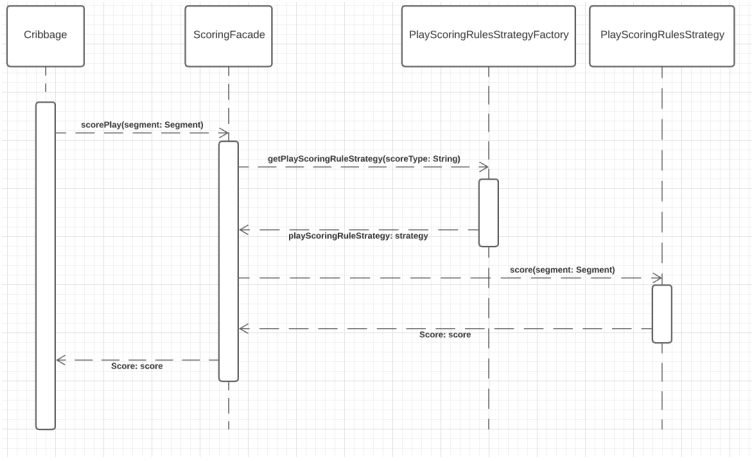
All four of the strategy pattern implementations in the scoring subsystem employ, as per the strategy pattern, Singleton Factories to instantiate the individual strategies being used, as well as handle the logic of determining which strategy or strategies are needed.

All four of the strategy pattern implementations also have an abstract, basic composite class that their respective concrete composite classes extend. Originally they were simply set up without concrete composite classes, instead just having a single normal composite class to handle the simultaneous occurrence of various scoring events (e.g. a fifteen and pair3 happening at the same time in the play phase). The design was altered to contain concrete composites extending abstract composites to maintain protected variation in the scoring subsystem, allowing for the implementation of new methods to handle the simultaneous occurrence of score events in the Cribbage round being played. This also allows for the ability to implement variant rules for the Cribbage round being played with minor alterations to the scoring subsystem to inform it which variant is being used.

The strategy patterns employed in the scoring subsystem also help with system polymorphism, reducing the reliance on complex conditional logic within the Cribbage class, as well as helping to maintain low coupling with the originally provided Cribbage Package.

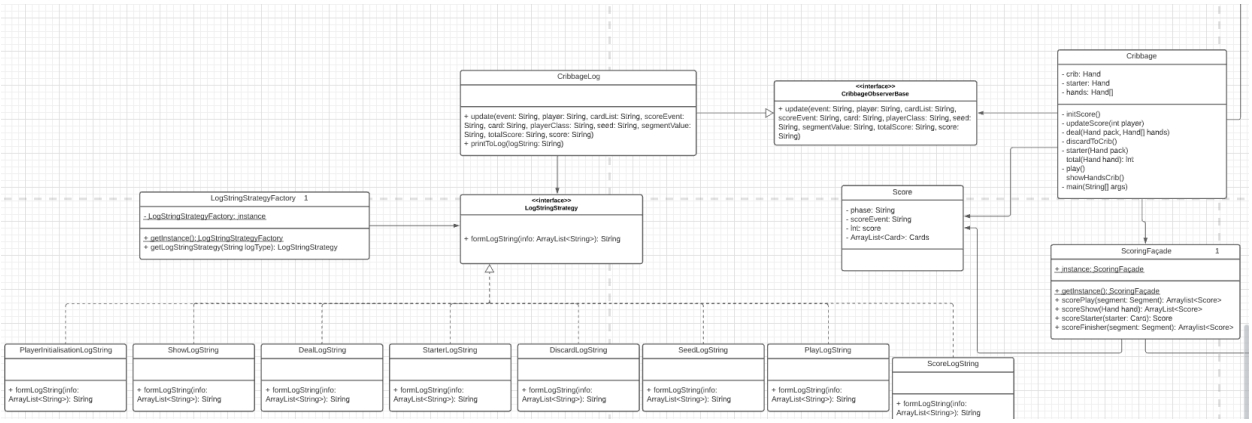
To support the logging system whilst maintaining high cohesion in the Cribbage class, a new Score class was implemented in the Cribbage package.
This class was created to simply act as a complex data type that stored the relevant information of a particular scoring event. Specifically, it stores the phase it corresponds to (currently, despite

having its own phase-defined set of strategies, the JackStarterStrategy is considered to occur in the play phase), the score increase from the score event, a string identifying the score event (e.g. fifteen, flush4, etc.), and a list of the cards that contributed to the score event.
The creation of this class allowed High Cohesion to be maintained in the Cribbage class by outsourcing the deduction of the form of scoring event occurring to the scoring subsystem, whilst also allowing this information to be easily passed to Cribbage so it could easily pass it to the logging system that was implemented. However, it does slightly increase the coupling between the Cribbage class and scoring system by requiring them to both respectively access or provide the same information. This increase is relatively minor though, and is outweighed by the higher cohesion it helps maintain, especially as the basic list of attributes of a scoring event are unlikely to require alteration in the future.



The basic operation of the scoring system can be seen in the Dynamic Design Diagram to the left. Do note that this diagram only covers the use of the playScoringRulesStrategy pattern, with the other two being omitted due to the messaging between classes being identical, just using some alternative methods and classes instead.

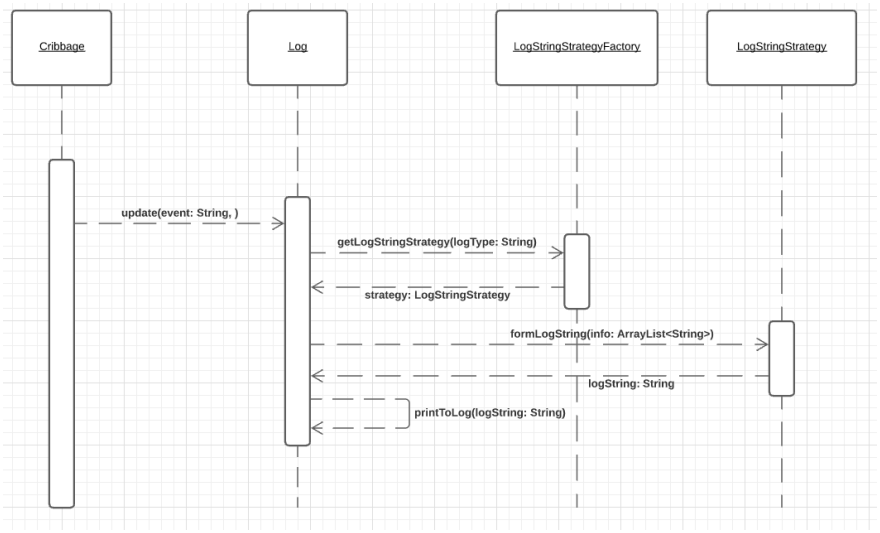
Logging System



The Logging System was implemented using the Observer Pattern as a basis. As Cribbage is the only class expected to publish anything, due to it handling the general operation of the round played, there is not an abstract Publisher class that Cribbage inherits from, since it is unlikely that more publisher classes would be added to the system.
The Observer class (CribbageLog) in the logging system inherits from an abstract observer class, as per the Observer pattern. However, despite the logging system being implemented within a package separate from the Cribbage package. This was done to ensure protected

variation by allowing for the implementation of observers that act in different ways to the logging system
CribbageLog then subscribes to Cribbage, and takes varying sets of strings published by Cribbage through the update function.

The Log Strings that are passed to the log file are constructed using a strategy pattern, with the specific strategy employed being determined by CribbageLog before it calls the factory to instantiate the appropriate strategy. This allows for low coupling with the actual logging process, as well as for the alteration of log string formats, or the introduction of new logging events, without doing much to the CribbageLog class.



The basic operation of the logging system is shown in the Dynamic Design Diagram to the left.

Additional Design Decisions

Some alterations to the provided Cribbage package were made, in order to support the implementation of the scoring and logging packages we added.
Many of the functions in Cribbage that were not public were made public, to allow the classes in the scoring and logging systems to employ the useful functionalities within the Cribbage class without having to re-implement those functions.
A getter method was also added to the Segment class, which returns the value of the go Boolean, for the purposes of the finisher scoring strategy.
Additionally, a new method called cardOrder was implemented in the Cribbage class to allow the scoring system to extract the order of each particular card, where ace = 1, jack = 11, queen = 12 and king = 13. This method was implemented specifically for the purpose of identifying sequences.

An alternative approach that was considered was to have both the Logging and Scoring Systems rely on the Observer Pattern indirectly, using an event observer that would call the Scoring and Logging subsystems as appropriate, determining which scoring strategies were appropriate based on the segment or hand, and constructing log strings for the logging system. This approach was discarded due to the low cohesion it created in the event observer class being employed, as well as the excessive complexity it created and the acknowledgement that Cribbage needs to actually know about the scoring subsystem, which goes against the basic concept of the Observer pattern. This approach also would have led to high coupling between

the log and scoring systems, as they would both rely on the same observer class, complicating any attempts to alter its behaviour.