SWEN30006 Software Modelling and Design Project 2: Pasur Trainer

Tute: Monday 3:15 - 5:15pm

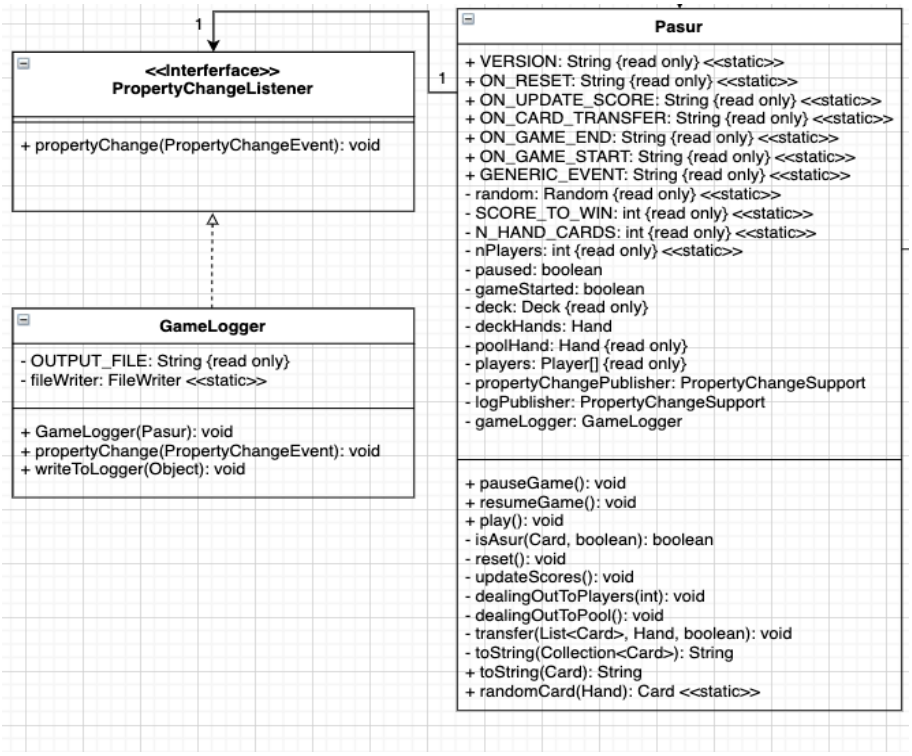
Finbar Howard - 1082205
James Hollingsworth - 915178
Oliver Bestel de Lezongard - 914956

Introduction

The team at NERD Games currently has a running version of a Pasur trainer. At a basic level, Pasur is a card game in which the purpose of the game is to score as many points as possible. Each game consists of 2 to 4 players, a pool, a deck, and a player's surs. The team's current implementation of the Pasur trainer does not provide any scoring or a log that keeps track of the run sessions. The focus of this project is to update this system to implement scoring and a logger. This design needs to be extensible and allow flexibility for future maintenance and modification. The purpose of this report is to justify the Gang of Four design patterns and GRASP principles to incorporate these two aspects while promoting an extensible design to allow for future maintenance and modification.

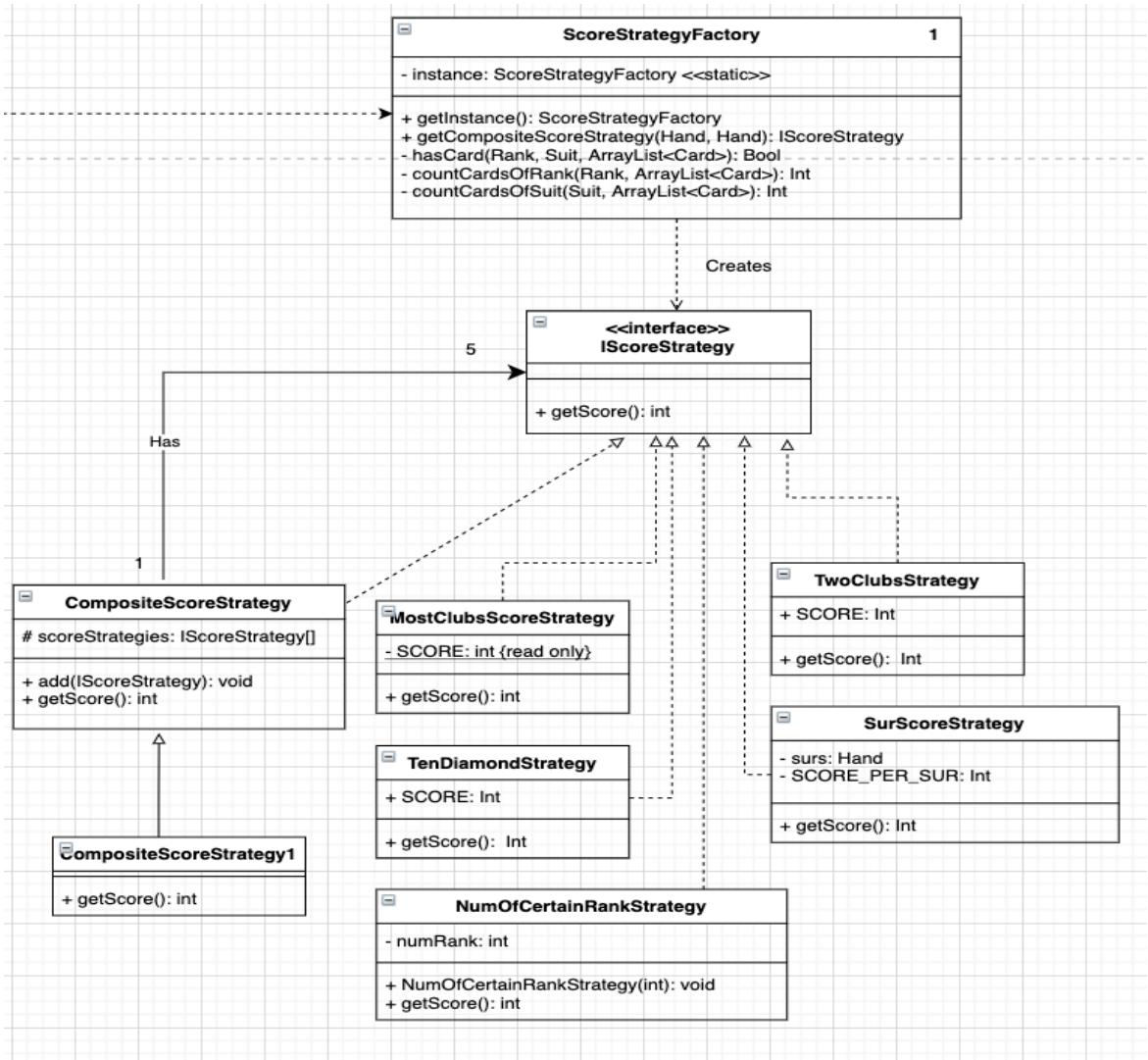
Implementation of Logging

The purpose of logging is to allow for players to study the impact of their choice on the play and the score. The log is dependent on events within the gamestate, and thus it is clear to see that the **Observer/Publish-Subscribe Pattern** is the most suitable in this case. The class that controls the log is a subscriber to the concrete Pasur publisher object, and provides an interpretation of the game events in the form of a line in the log when the Pasur class generates and publishes a relevant event. This pattern implements a 'listener' interface as illustrated in the diagram below. We can identify this point as a possible predicted variation point due to the possibility of further subscriber objects being added. These objects may have their own unique reactions to published Pasur game events being added in the future. Thus our implementation of the **Observer Pattern** uses the GRASP principle of **Protected Variations**. This implementation of **Protected Variations** makes use of **Indirection** and **Polymorphism** to allow for future implementation of pluggable 'subscriber' objects,to handle alternative 'subscriber' objects based on type, and to provide protection around a point of predicted variation. This design choice results in an extensible design for 'subscriber' objects such as a game log, while simultaneously promoting **Low Coupling**.

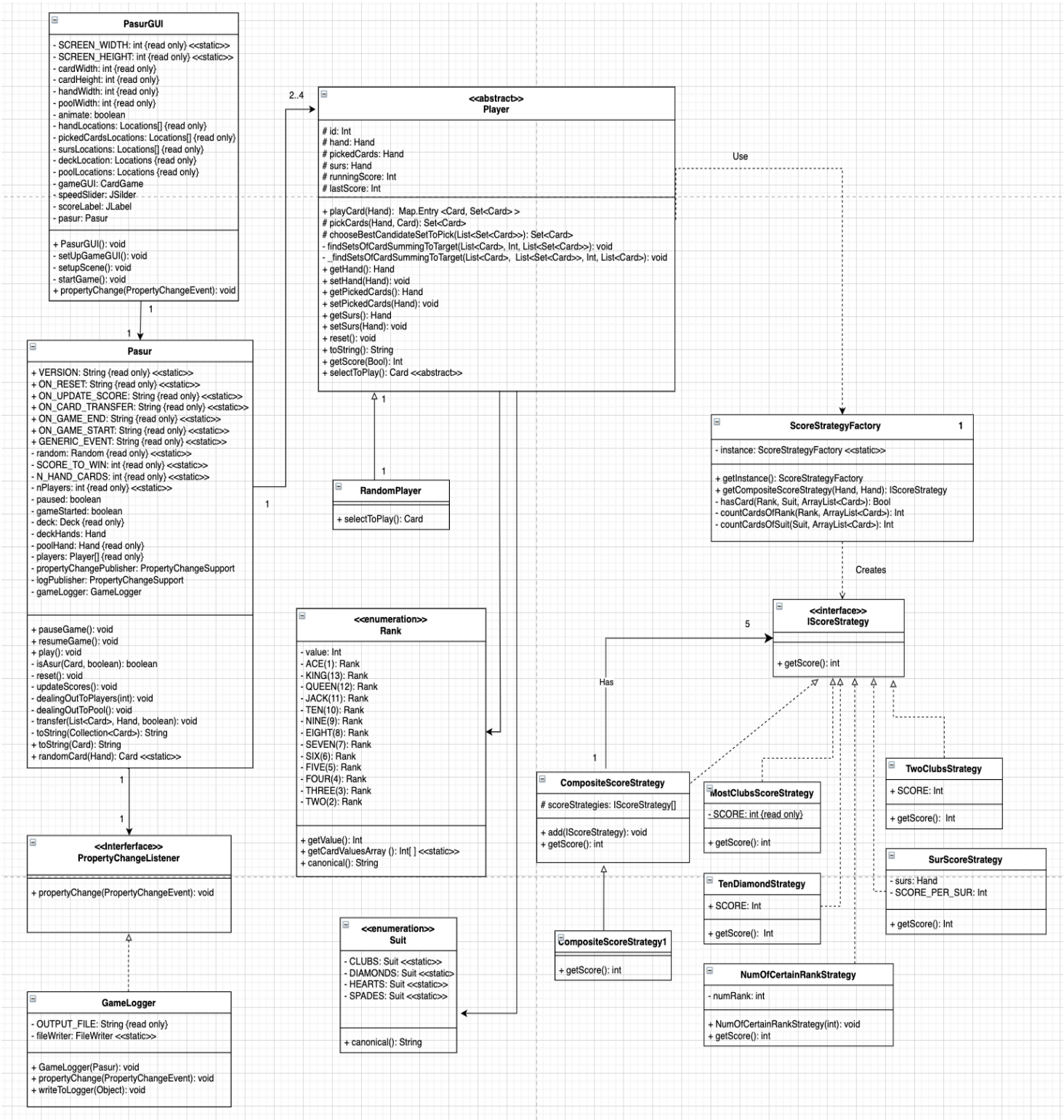


Implementation of Scoring

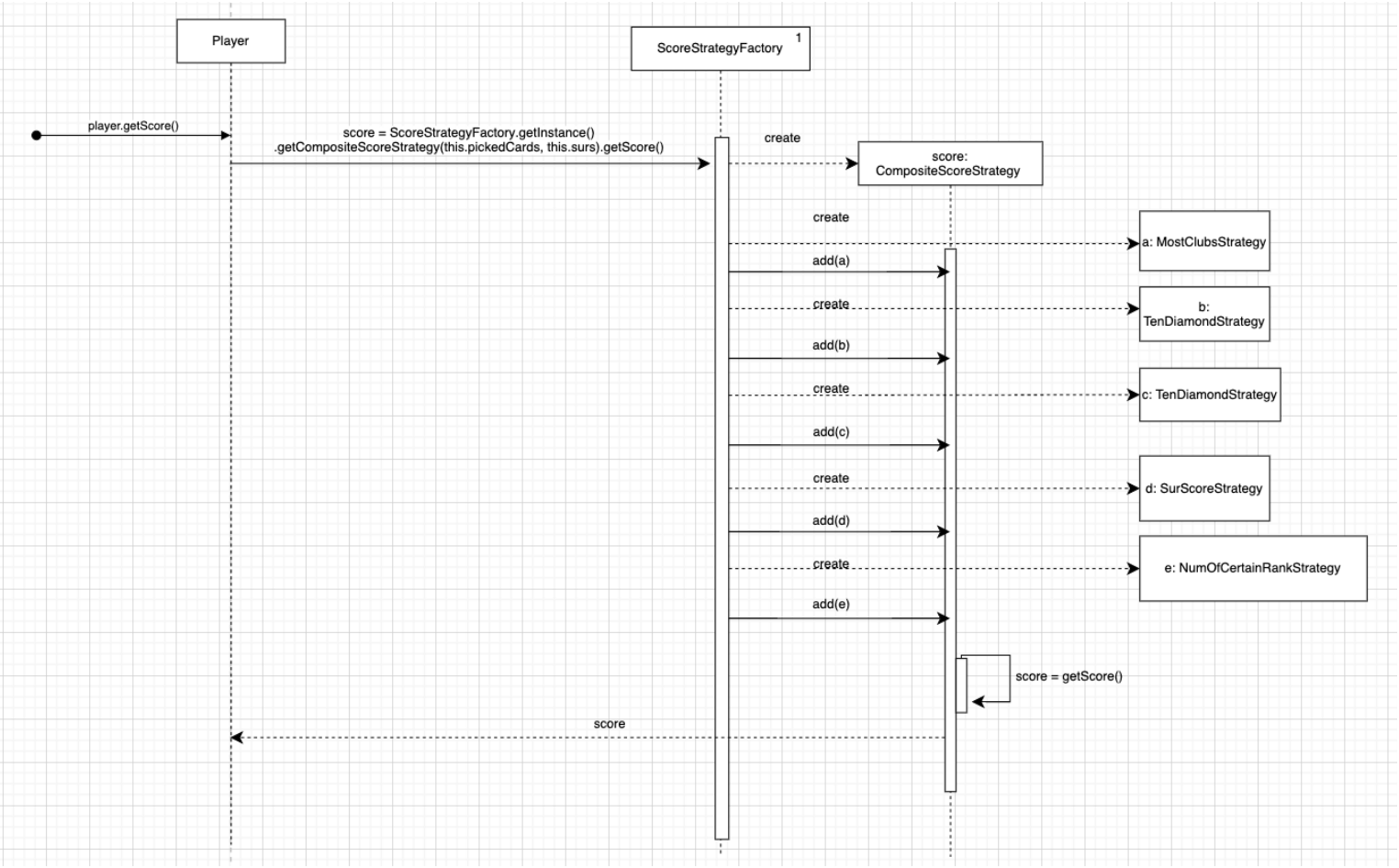
The implementation of a scoring system into the Pasur game has to be considered carefully. It is clear that this is a potential point of extension and evolution, the scoring rules of Pasur are not universal and may need to be changed or adjusted. Alongside this we have many different rules that go into scoring Pasur that all need to be accounted for and handled without causing high coupling. To handle the point of extension for rules we implemented a **Strategy Pattern.** This IScoreStrategy interface allows us to create multiple scoring strategies that implement the strategy pattern. The strategy pattern is an example of the GRASP principles of **Polymorphism** and **Protected Variation** which allows us to create pluggable components to handle any new game rules with a stable interface around a point of known variation. In future if there needs to be changes or additions to the scoring system these can be added as implementations of the base strategy pattern promoting **Low Coupling**. All strategies share a getScore() method which is why an **Adapter Pattern has not been used**. To handle the creation of these different strategies we have included a **Factory Pattern**, this **Pure Fabrication** allows us to have a dedicated class to create all strategies to promote **High Cohesion** and **Low Coupling**. This factory is created as a **Singleton** as we need a global and single point of creation and contact for all strategies. All of these strategies must be implemented together in the scoring system, to create greater cohesion we have used a **Composite Pattern**. This pattern allows all of our strategies to be implemented together through one centralised class. This CompositeScoreStrategy implements the IScoreStrategy to effectively score using all strategies that implement the interface. This provides **High Cohesion** and allows for easier variation as any new strategy that is added for scoring will be incorporated via the CompositeScoreStrategy object.



Design Class Diagram



Design Sequence Diagram for Computing Score



Conclusion

In conclusion, regarding the implementation of a log, we have implemented the **Observer Pattern**. This makes use of **Protected Variations** with **Polymorphism** and **Indirection** to allow future implementation of pluggable 'subscriber' objects,to handle alternative 'subscriber' objects based on type, and to provide protection around a point of predicted variation. For the implementation of scoring, we have used the **Strategy, Singleton,** and **Composite Patterns**, which make use of a **Singleton Factory, Protected Variations, Polymorphism, and Pure Fabrication** to allow for complex rule-based scoring logic to be implemented for the Pasur game, while simultaneously allowing us to create pluggable components to handle any new game rules with a stable interface around a point of known variation. These design choices encourage **High Cohesion** and **Low Coupling**, while providing an extensible design that is flexible to future modification and maintenance.