DESIGN PATTERN

Ricochet Robots Project

Group #6:

McDonald, Brianna
Juwaheer, Yudish
Younis, Faisal
Chikwati, Ngoni Nigel

Design Patterns:

While designing and implementing our version of Ricochet Robots throughout the semester, we tried to keep several key design patterns in mind. One example of a pattern we used when implementing our game is the Creator pattern. For instance, our StartFrame class gets information from the user about the number of players, which game board to use, whether or not to enable colour vision assistance, and so on. Therefore, it made sense to assign the responsibility of creating SimpleMap and ComplexMap objects to the StartFrame class, since its main job is to record the information used for creating instances of those classes. The initializing data that StartFrame contains is then directly passed to the SimpleMap and ComplexMap objects upon their creation. The advantage of this pattern is that it leads to low coupling, since, otherwise, other classes would need to get information from StartFrame in order to create the game boards, and also higher opportunity for reuse, since the StartFrame class can be easily modified and used in other systems.

Another design pattern used in our system in several places is the High Cohesion pattern. For example, we have several lightweight classes that have specific responsibilities and that can be easily reused for other systems. These classes include the Help class and the About class, which are responsible for displaying a window with instructions or information about the game. The Load class also displays high cohesion since its only responsibility is to load saved game data. Due to the high level of cohesion, these classes are very easy to understand and also contribute to low coupling in the system.

To further add to our use of high cohesion and low coupling, a design pattern that would have been good to implement if we had more time is the Controller pattern. This pattern could have been used to create classes such as SimpleMapController and RobotController that would handle the robot and game board input system events. The Controller pattern is used to separate the application logic from the domain objects and to make the code which provides these object's functionality much clearer. In this case, the use of this pattern would have made the domain classes such as SimpleMap and Robot much more highly cohesive and helped with maintaining low coupling. These patterns are extremely beneficial as they make code easier to reuse and help to make it so that changes in one part of the system lead to minimal or no changes in other parts.