Assignment 3: Exceptions, Design Patterns

Q2-b) Recovery Decisions for Exceptions

RobustArray::at

FlexArray originally throws out_of_range exception for at. RobustArray catches the exception. RobustArray calls the FlexArray’s at function, and returns the array’s default value. Since returning an exception is not that stable, and there is no other value more stable than default, therefore the default value as the successful return for RobustArray’s at function is used.

RobustArray::assign

FlexArray originally throws out_of_range exception for assign. RobustArray catches the exception. RobustArray calls the FlexArray’s assign function and when the out_of_range exception is encountered. It would not do anything. One alternative may be growing the size of the array to the proposed index position, and assign the value there. However, when there is not enough memory to allocate for the resize function, RobustArray would have to use the recovery of the resize function. This may cause results that the users do not expect. Therefore, doing nothing when out_of_range exception is encountered would be the ideal recovery solution.

RobustArray::resize

FlexArray originally throws bad_alloc exception for resize. RobustArray catches the bad_alloc exception and attempt at maximum twice, to resize the array to smaller-than-requested size. The first time is half of the extra size, and second time is a quarter of the extra size. Cutting the extra size by half each time is reasonable as it is more balanced. After two attempts, the operation throws BAException, which is derived from std::bad_alloc, and it stores the value of the size used in the latest attempt.

Q3

a) GameCompany is writing a new expansion to a game that upgrades the existing enemy AI to allow for dynamic generation of different enemy types. An abstract base class for the AI and a concrete AI class already exists. As an ongoing commitment to enhanced game play, GameCompany intends to implement additional AI behaviours on a regular basis. The decorator pattern will be used to implement this feature.
b) The following is the UML model with the application of the decorator design pattern. Elements with a white background highlights the new additions to the model while the elements with a grey background indicates the original components of the model. The ActionDecorator abstract decorator class allows the concrete decorators, such as YellDecorator, ShootDecorator and DodgeDecorator to wrap around the BaseAI class and provide additional AI behaviours.

Note that ActionDecorator doesn't need to specify private data member AI* - this data member already modelled, using aggregation.