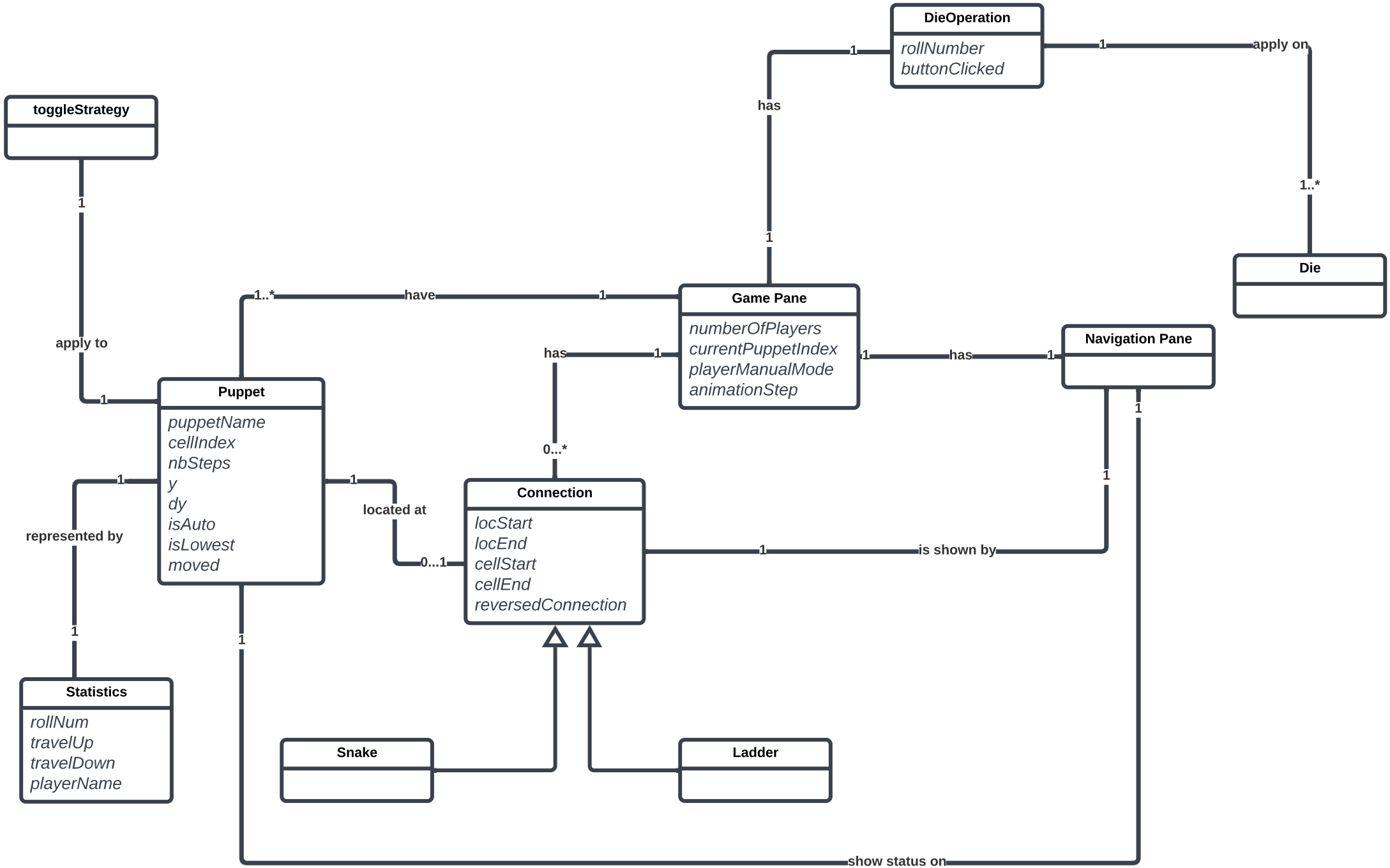
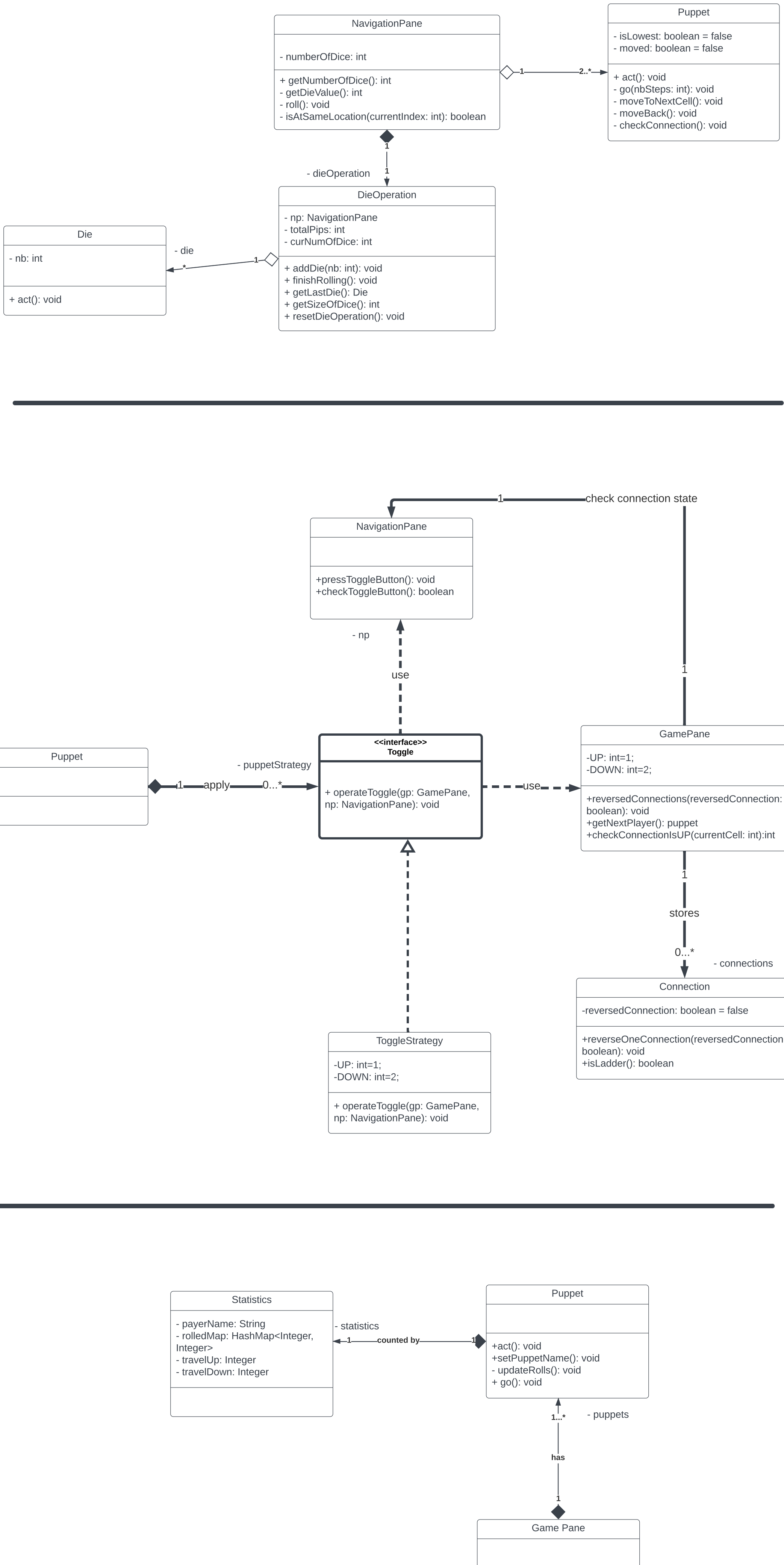
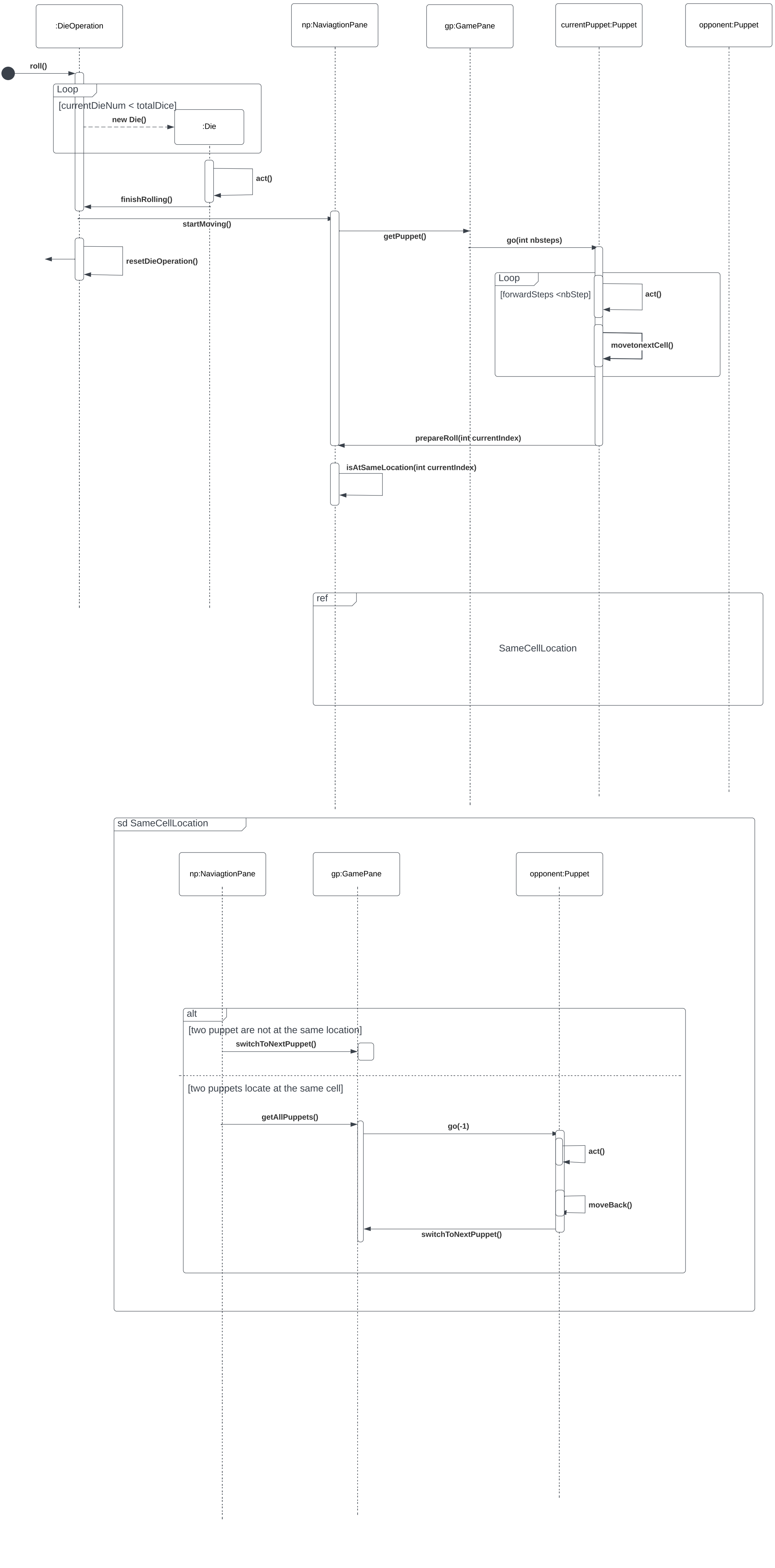

DieOperation
rollNumber
buttonClicked
toggleStrategy
Die
Game Pane
numberOfPlayers
currentPuppetIndex
playerManualMode
animationStep
Navigation Pane
Puppet
puppetName
cellIndex
nbSteps
y
dy
isAuto
isLowest
moved
Connection
locStart
locEnd
cellStart
cellEnd
reversedConnection
Statistics
rollNum
travelUp
travelDown
playerName
Snake
Ladder
1
apply on
1..*
has
have
apply to
located at
0...1
0...*
represented by
is shown by
show status on


NavigationPane
- numberOfDice: int
+ getNumberOfDice(): int
- getDieValue(): int
- roll(): void
- isAtSameLocation(currentIndex: int): boolean
Puppet
- isLowest: boolean = false
- moved: boolean = false
+ act(): void
- go(nbSteps: int): void
- moveToNextCell(): void
- moveBack(): void
- checkConnection(): void
1
2..*
1
- dieOperation
1
Die
- nb: int
+ act(): void
- die
*
1
DieOperation
- np: NavigationPane
- totalPips: int
- curNumOfDice: int
+ addDie(nb: int): void
+ finishRolling(): void
+ getLastDie(): Die
+ getSizeOfDice(): int
+ resetDieOperation(): void
1
check connection state
NavigationPane
+pressToggleButton(): void
+checkToggleButton(): boolean
- np
use
1
Puppet
1
apply
0...*
- puppetStrategy
<<interface>>
Toggle
+ operateToggle(gp: GamePane, np: NavigationPane): void
use
GamePane
-UP: int=1;
-DOWN: int=2;
+reversedConnections(reversedConnection: boolean): void
+getNextPlayer(): puppet
+checkConnectionIsUP(currentCell: int):int
1
stores
0...*
- connections
Connection
-reversedConnection: boolean = false
+reverseOneConnection(reversedConnection: boolean): void
+isLadder(): boolean
ToggleStrategy
-UP: int=1;
-DOWN: int=2;
+ operateToggle(gp: GamePane, np: NavigationPane): void
Statistics
- payerName: String
- rolledMap: HashMap<Integer, Integer>
- travelUp: Integer
- travelDown: Integer
- statistics
1
counted by
1
Puppet
+act(): void
+setPuppetName(): void
- updateRolls(): void
+ go(): void
1...*
- puppets
has
1
Game Pane
getStatistics(): LinkedList<Statistics>

A full turn invloving a player landing one the same square as the other player and forcing the other player back onto a downward path symbol



Task 1

The NavigationPane class handles the input and output of the game which acts as a controller class. However, the controller class cannot handle too many operations, which will lower the cohesion. For task 1, we might need more than one die in a game. In this case, if we directly put the action of rolling a list of dice into ***NavigationPane*** class, the ***NavigationPane*** class is probably doing too much work. In order to avoid this, we have increased the cohesion by creating a ***DieOperation*** class. This is the GRASP-pure fabrication method where the ***DieOperation*** class encapsulates the responsibilities of rolling the dice and getting the total pips. After the last die is rolled, ***DieOperation*** class calls the function to stimulate the movement of the puppet. Moreover, the ***DieOperation*** class is also responsible for creating ***Die***, which is stored as an array list.

Task 2&3

For task 2, we added an attribute "isLowest" to detect if the puppet rolls the lowest possible number in the ***Puppet*** class. For task 3, we firstly add a function in ***NavigationPane*** class to detect if two puppets are located in the same cell after one puppet finished its turn. Then, we call the *go()* function in ***Puppet*** class to move the other puppet one step back. We also add an attribute to detect if the puppet has moved. This will notice the puppet to move backwards instead of move forwards until the number of steps becomes less than 0. These are based on the knowledge of information experts. The ***Puppet*** class can manage if it needs to travel down when they roll the possible lowest number. Also, it needs to contain the information if it is asked to move backwards.

Task 4

In task 4, it is asked to apply a strategy which allows all the connections to be reversed in both auto mode and manual mode. So, we use GRASP-protected variation methods to create an interface class ***"Toggle"***. The advantage of PV here is that it achieves the Open-Closed Principle where the strategy can be opened for extension and closed for modifications. In other words, the ***Toggle*** interface allows different strategies to be applied on reversing connections, but it won't give the permissions to other people to modify the pre-written strategy. In this task, we only need to achieve reversing connections by a simple strategy, so we decided to create a class ***"ToggleStrategy"*** which implements the ***Toggle*** interface. In the future, if there are more different strategies, it is very easy to create another class and implement the ***Toggle*** interface, which is the idea of Polymorphism. Then, we decided to make an association between ***ToggleStrategy*** and ***GamePane***. ***GamePane*** stores the arraylist of connections which can easily call the reversing connection method under the ***Connection***

class, rather than calling the reversing all of the connections method directly in ***ToggleStrategy.*** This design would maintain the low coupling and high cohesion.
Next, we looked through all the classes, and we were able to figure out that ***Puppet*** closely uses the ***ToggleStrategy*** because the toggle strategy is only used under the auto mode and the auto/manual state is set/changed in the ***Puppet*** class. So, ***Puppet*** and ***ToggleStrategy*** is a GRASP-Creator relationship where the ***Puppet*** class creates ***ToggleStrategy*** class.

Task 5
Firstly, we designed a ***Statistics*** class by using a pure fabrication method. If we want to add more statistics in the future, we only need to modify the ***Statistics*** and ***Puppet*** class. This gives us a lower coupling. Then we have a *getStatistics()* method in GamePane so that GP can ask all the puppets for their own statistics data.
In terms of implementation, the puppet could know whether it should go up or down in the *act()* method, and therefore it can update the number of travelUp and travelDown attributes in *Statistics* class. Apart from that, the number of rollMap in ***Statistics*** can be updated before the ***Puppet*** moves in the *go()* method.