Workshop Friday 13:00, Team 8

SWEN30006 Report - Project 1 Snakes And Ladders

Part 1: From Problem Domain to Domain Model

To gain a solid understanding of the problem domain a Domain Class model (see Figure 1) was constructed to map out the required modifications to the Snakes and Ladders Project.

Part 2: From Domain Model to Design Model

The Design Class Diagram (see Figure 2) accurately reflects the alterations made to the existing SLOP model in order to accommodate changes concerning rules and gameplay according to the design specification. The sections below will introduce more detailed information concerning the modification and justification of the design.

Part 3: Design Sequence for Same Square as Opponent

A design sequence diagram (see Figure 3) was created to help visualise the methods needed to model the behaviour where a player pushes an opponent backwards if they both end up on the same landing tile.

Task 1: Preset Number of Dice

The function getDieValue() was modified from the NavigationPane class to be able to take multiple dice by storing in an array of diceValues. It was altered to properly take in values from the properties file, based on the number of dice in the game. Additionally, within the same class, roll() was edited in order to accept the new output of getDieValue(), which now returns an integer array instead of an integer. This reduces coupling between the Die and NavigationPane classes.

Inside the Die class, getters and setters were added to get the total dice roll each time a roll is performed. This was implemented into the roll() function to use that total, thus allowing the players to move the total rolled amongst the available dice.

Task 2: Lowest Roll and Travelling Down a Path

Within the Puppet class, the act() function's section which pertains to the detection of a connection was altered. This was done in order to initially verify whether the connection was travelling downwards, and if so, then examines if the accumulated

total rolled is equal to the number of dice. By using this method, the Snakes and Ladders Framework is then able to infer the ‘lowest possible roll’.
Should such conditions be fulfilled, a descent made by the player shall be prevented. These conditions then call for functions that would end further player movement, as consistent with landing on a tile with no connections.

Task 3: Same Square as Opponent

The prepareRoll() function was altered within the NavigationPane class to identify situations where a player would land on the same square as their opponent, creating a situation where the function go() would have a go(-1) state.

Finally, the rest of the changes were made within the Puppet class where the existing function moveToNextCell() was copied and used as a base to create a movement function moveToPrevCell() which pushes a player back to the previous cell. Following the creator pattern, contraindications were used to recycle objects of significant complexity. A boolean variable called goBack was created and initially set to False to be used as a checker within the act() function. The purpose of goBack is to ensure that the player does not go back into the round and have another turn once it is pushed back.

After the modifications within NavigationPane, there is now a situation within the Puppet class where the go() function can have a state of go(-1). Thus the act() function is further altered under the ‘normal movement’ section loop to factor in the behaviour where the opponent gets pushed back if landing on the same square as the player. This is done by utilising the newly made moveToPrevCell() function and incrementing nbSteps accordingly. Polymorphism is used as both the Die and the Puppet class inherit from the Actor class to show the sprites moving.

Task 4: Reverse Button and Player Strategy

The toggleable button to reverse the path of both snakes and ladders on the whole board was created by making a reverseConnection() function with the class Connection. This function swaps round the instance’s attributes of cellStart and cellEnd making use of a temp attribute as a placeholder. Through this swapping, it recalibrates the attributes of locStart and locEnd (used for sprites). As reflected in the Design Class Diagram (see Figure 2), the Connection class aggregates both the Snake and Ladder classes to achieve a low representation gap following the Creator pattern as well as using polymorphism for Connection to inherit Snake and Ladder.

A getter for all connections was set within the GamePane class and the integration of the toggle button itself was done within the NavigationPane class. Here polymorphism was utilised for GamePane and NavigationPane to inherit from GameGrid as a method to achieve protection at a variation point and decrease

coupling. Within NavigationPane, a boolean checker attribute called isToggle was utilised as a flag to indicate the status of the toggle button, and to flip all connections via the GamePane getter function if isToggle is True. This is checked at the start of each turn under the prepareRoll() function.

Here logic was implemented for each player to reverse the connections based on the current locations of opposing players within the Puppet class. By looking at the current locations of all opposing players it considers all possible rolls that each opposing player can make within their turn. It first detects if an opposing player can land on a connection, then checks whether that connection takes the player up or down the board. Either possibility is added to the cumulative total of the attributes numUps and numDowns. If the total number of numUps are greater or equal to the number of numDowns, than a strategy will be employed and the player will reverse the connections.

Task 5: Basic Statistics

In order to gain access to tracking the basic statistics of each player, two new attributes were implemented to the Puppet class - both being hash tables - diceStatitics and pathStatistics. Both hash tables were initialised upon instantiation to contain zero-counter values.

To gain information, diceStatistics and pathStatistics were both updated in accordance to player interactions. The former relating to total dice rolls and the latter in regards to connection movements (Note: movements prevented through Clause 2 are ignored). Correspondingly, output functions printDiceStats() and printPathStats() were created for the Puppet class to print these statistics in accordance with the project-specified format. These are called from the NavigationPane class for all existing players upon the conclusion of the game.

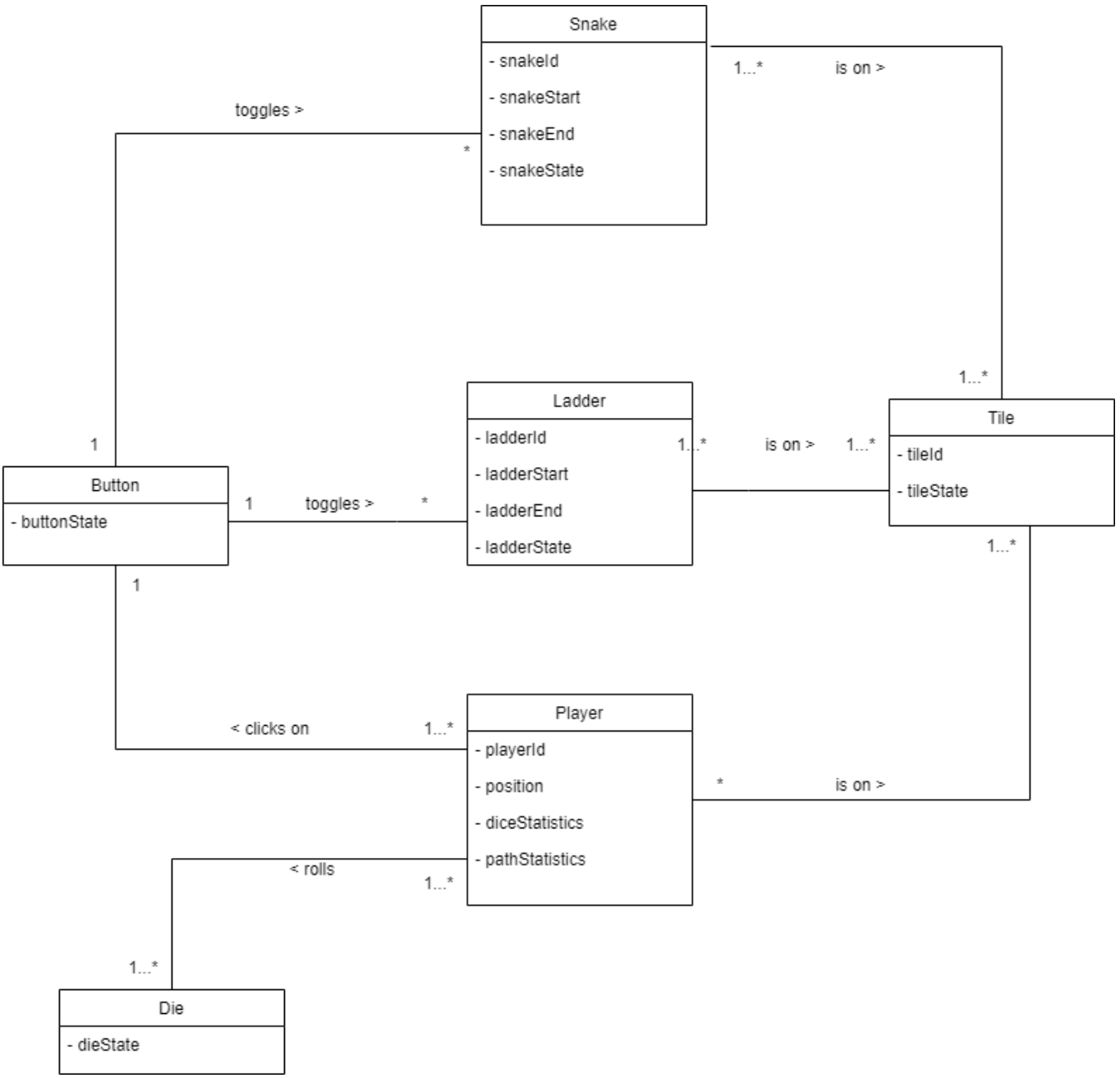
Snake
- snakeId
- snakeStart
- snakeEnd
- snakeState
1...*
is on >
toggles >
*
1...*
Ladder
- ladderId
- ladderStart
- ladderEnd
- ladderState
1...*
is on >
1...*
Tile
- tileId
- tileState
1
Button
- buttonState
1
toggles >
*
1...*
1
< clicks on
1...*
Player
- playerId
- position
- diceStatistics
- pathStatistics
*
is on >
< rolls
1...*
1...*
Die
- dieState

Figure 1

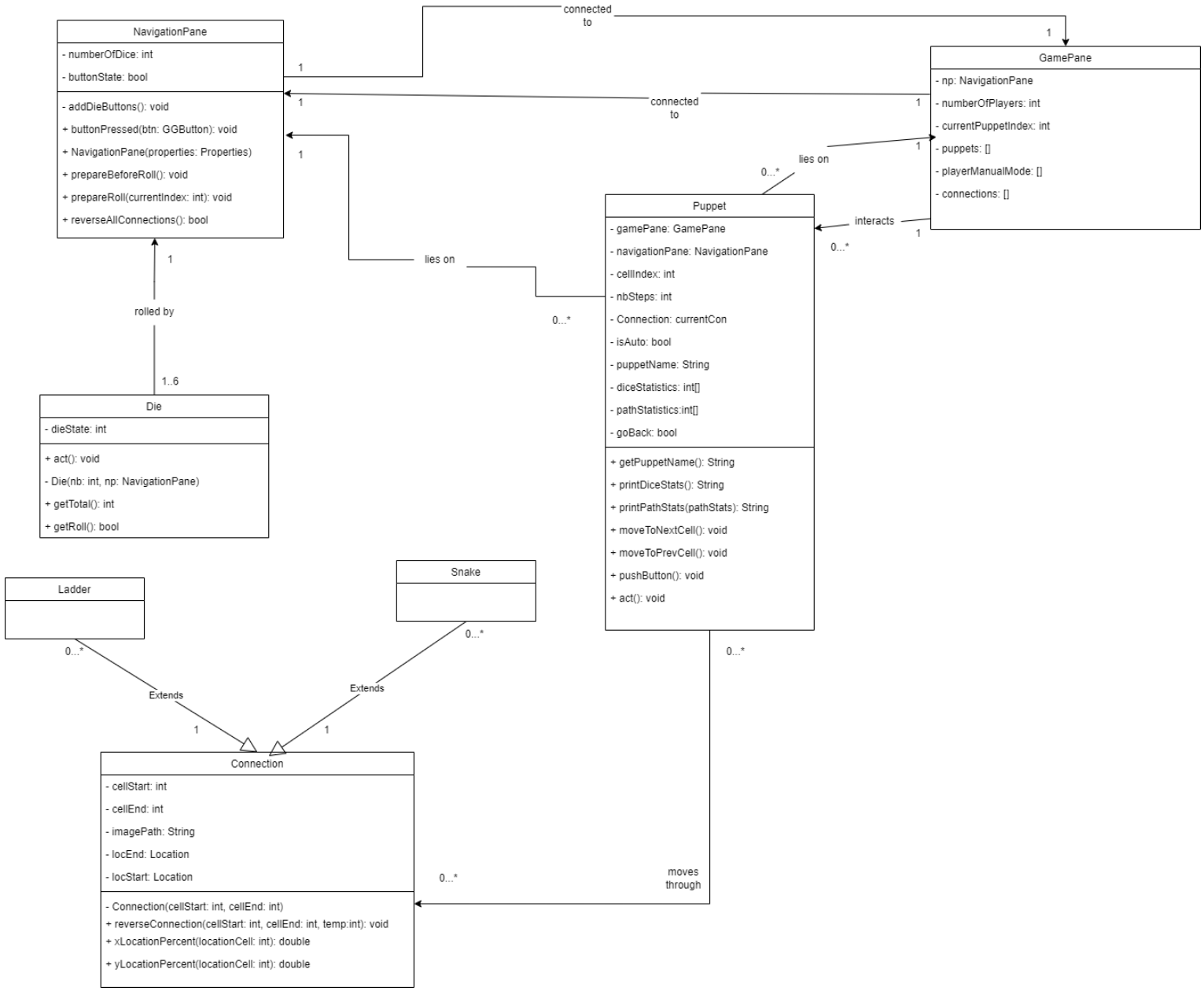
NavigationPane
- numberOfDice: int
- buttonState: bool
- addDieButtons(): void
+ buttonPressed(btn: GGButton): void
+ NavigationPane(properties: Properties)
+ prepareBeforeRoll(): void
+ prepareRoll(currentIndex: int): void
+ reverseAllConnections(): bool
GamePane
- np: NavigationPane
- numberOfPlayers: int
- currentPuppetIndex: int
- puppets: []
- playerManualMode: []
- connections: []
Puppet
- gamePane: GamePane
- navigationPane: NavigationPane
- cellIndex: int
- nbSteps: int
- Connection: currentCon
- isAuto: bool
- puppetName: String
- diceStatistics: int[]
- pathStatistics:int[]
- goBack: bool
+ getPuppetName(): String
+ printDiceStats(): String
+ printPathStats(pathStats): String
+ moveToNextCell(): void
+ moveToPrevCell(): void
+ pushButton(): void
+ act(): void
Die
- dieState: int
+ act(): void
- Die(nb: int, np: NavigationPane)
+ getTotal(): int
+ getRoll(): bool
Ladder
Snake
Connection
- cellStart: int
- cellEnd: int
- imagePath: String
- locEnd: Location
- locStart: Location
- Connection(cellStart: int, cellEnd: int)
+ reverseConnection(cellStart: int, cellEnd: int, temp:int): void
+ xLocationPercent(locationCell: int): double
+ yLocationPercent(locationCell: int): double
connected to
connected to
lies on
interacts
lies on
rolled by
Extends
Extends
moves through
1
0...*
1..6

Figure 2

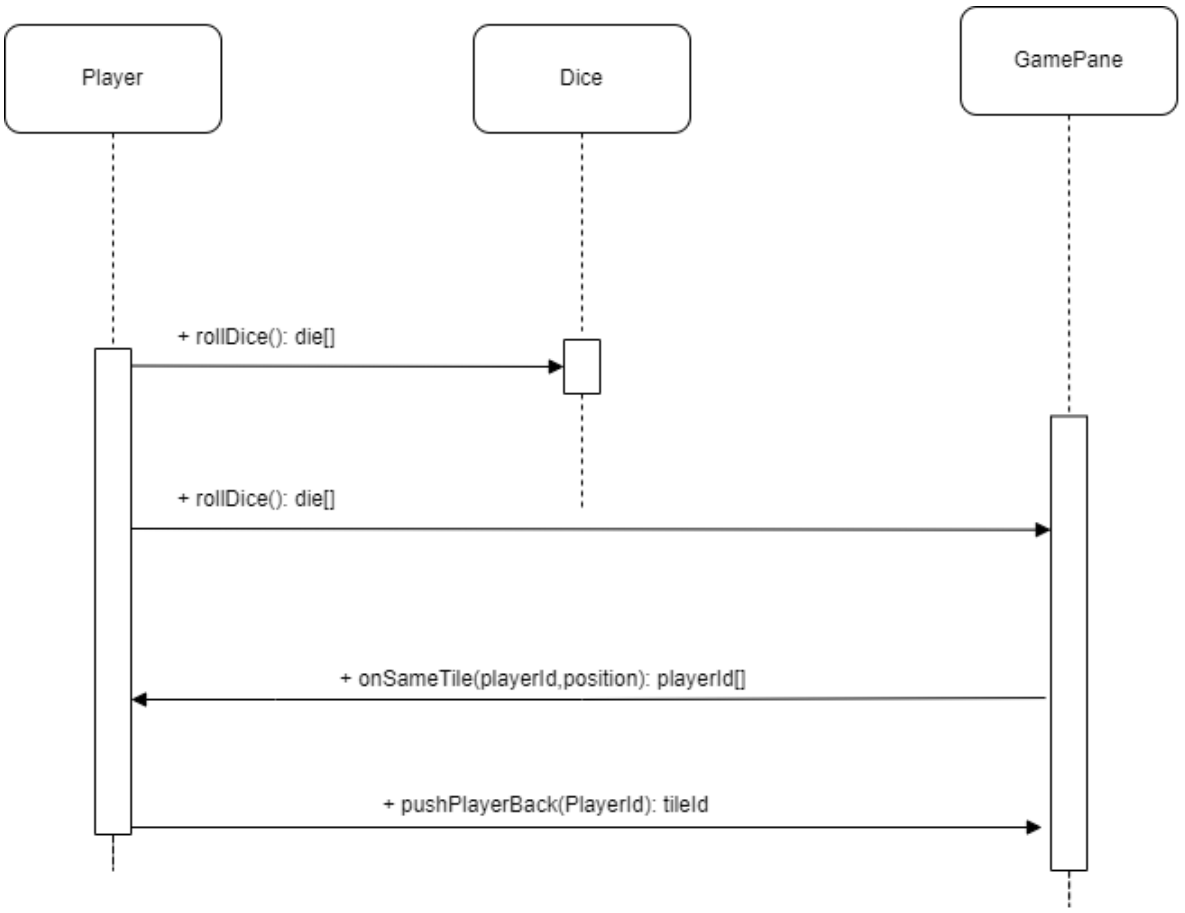
Player
Dice
GamePane
+ rollDice(): die[]
+ rollDice(): die[]
+ onSameTile(playerId,position): playerId[]
+ pushPlayerBack(PlayerId): tileId

Figure 3