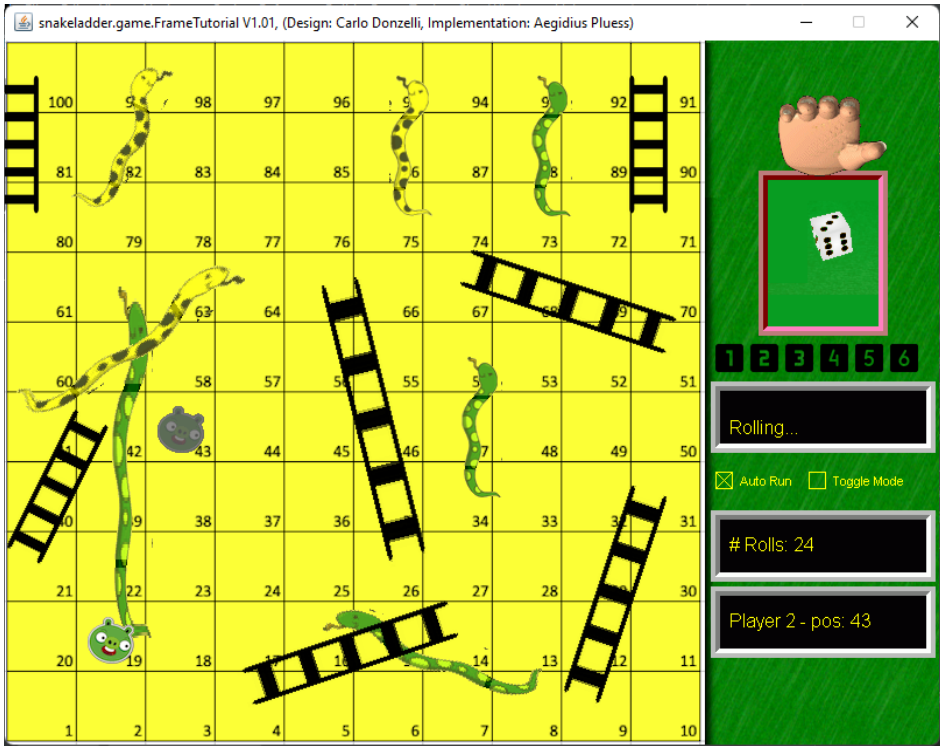
SWEN30006 Software Modelling and Design
Project 1: Snakes and Ladders (due: Wed Apr 27th 11:59pm)
- Project Specification -

School of Computing and Information Systems
University of Melbourne
Semester 1, 2022

Background: Snakes and Ladders

The Games division of *New Exciting Realtime Discoveries Inc. (NERDI)* has recently developed a video game based on a popular board game. NERDI Games hopes to enhance the game and turn the game into its' next big hit (also its' first big hit). The board game they have focussed on is Snakes and Ladders. They have already developed a vanilla digital version of the board game; NERDI marketing have called the digital version *Snakes and Ladders on a Plane* (or *SLOP* for short). NERDI need your help to make SLOP more interesting.



snakeladder.game.FrameTutorial V1.01, (Design: Carlo Donzelli, Implementation: Aegidius Pluess)

Rolling...

☒ Auto Run □ Toggle Mode

Rolls: 24

Player 2 - pos: 43

Figure 1: SLOP: Snakes and Ladders on a Plane

Definitions

- Turn: The full sequence of actions which starts with a player rolling the dice, then moving their piece and so on until it is their opponents turn.
- Move: Moving a player's token from one square to another, as a result of a dice roll, landing on a snake or ladder, or other rule of the game
- Land: A player lands on a square as a result of a move; rules relating to the square on which the player then landed must then be followed.
- Path symbol: a snake or ladder which provides a path between two squares on the board.

Your Task

The current version of SLOP seems to work reasonably well. However, the design and implementation of SLOP are not well documented. Your task is to make the following changes.

1. Allow for the game to use a preset number of (six-sided) dice, defaulting to one. The property file already contains the setting, but it is not currently used. Each dice should be rolled (randomly generated) in turn.
2. Players do not travel down a path symbol if they rolled the lowest possible roll (a 1 if using only one dice, a 3 for three dice and so on) to land on the symbol square.
3. If a player lands on the same square as their opponent when moving after rolling the dice (only), the opponent moves one square backwards and must follow the rules of the landing square.
4. The final step of a player's turn gives the player the opportunity to reverse the current roles of the path symbols, that is, if snakes were down and ladders were up, they can choose to make snakes up and ladders down.
    a. Snakes start as down paths and ladders start as up paths.
    b. An indicator is provided to show which way the path symbols operate (in the current version of SLOP, always snakes down and ladders up). This indicator must be set correctly.
    c. A button is provided to allow this setting to be toggled interactively.
    d. For the purpose of automated testing, each auto player will employ a simple strategy to decide whether to toggle: they will look at their opponent's position and where they could land as a result of their next dice roll – if there are at least as many possible landing squares with an upwards path symbol start as there are landings with a down path symbol start, the player will make the switch. *Note that NERDI games will want to change this strategy in the future – your design should consider how easy it will be to change this.*
5. In order to track game play, the game should keep some basic statistics: for each player, we want to know how many times that player rolled each possible value of the dice, and how many path symbols they traversed up and how many path symbols they traversed down. *NERDI will want to add additional statistics in the future.*
    a. An example of the required format for output of dice roll counts is:
       "Player 2 rolled: 2-7, 3-11, 4-0, 5-0, 6-2, 7-3, 8-1, 9-0, 10-1, 11-1, 12-2"
    b. An example of the required format of output for path symbol traversals is:
       "Player 1 traversed: up-3, down-7"

You need to apply your software engineering and patterns/principles knowledge to refactor and extend the system to support this new behaviour. It's recommended that you understand the high-level design of current system first so that you can effectively identify & update relevant parts. A partial, automatically generated class diagram for SLOP is included at the end of this specification.

In making the required changes, you can also change the existing design (and correspondingly, the implementation), however you like (and can justify in your report) but *you should not change* the way the program is run (main method, command line, and property files) or the existing text output. You do not need to change any aspects of the system which do not relate to the changed functionality above. You are not expected to make changes to the graphical elements of SLOP.

Hint: *Start with understanding the existing design at a high level, and with creating the domain class diagram. These will help you get an overview of the various elements involved and their relationships. You will need to generate and evaluate design options to make good choices and to justify your design for your design report.*

The Base Package

You have been provided with an Eclipse project export representing the current version of SLOP, including an example configuration files.

To begin:

1. Import the project zip file as you did for Workshop 1.
2. Try running by right clicking on the project and selecting **Run as... Java Application**: the entry point is *"snakeladder.game.FrameTutorial.main()"*.
3. As well as the SLOP GUI, you should see output in the Eclipse console showing you the current behaviour of the SLOP game.

This code should needs to be used as the starting point. Please carefully study the sample package and ensure that you are confident you understand how it is set up and functions before continuing. If you have any questions, please make use of the discussion board or ask your tutor directly in the workshop as soon as possible; it is assumed that you will be comfortable with the package provided.

Note: The SLOP property files allow you to specify player dice rolls to make it easier to test specific behaviour. Take a look at how this is set up and use it to your advantage.

Testing Your Solution
We will be testing your application programmatically, so we need to be able to build and run your program without using an integrated development environment.

The entry point must remain as "snakeladder.game.FrameTutorial.main()". You must not change the names of properties in the provided property file or require the presence of additional properties. If you add properties, they need to have defaults values as we will not set these in testing your submission.

Note: It is **your team's responsibility** to ensure that the team has thoroughly tested their software before submission.

Configuration and Project Deliverables
(1) Extended SLOP: Source code for SLOP extended according to the instructions above. You should not include property files or icons; we will provide these for test as per the original SLOP package which is why your extended SLOP should not depend on modified versions of property files or icons.

(2) Report: In addition to the extended SLOP, NERDI requires you to provide a report to document your design changes and the justification for your design. Details requirements for the report are available on the LMS submission page.

Submission
Detailed submission instructions will be posted on the LMS. *You must include your team number in all your pdf submissions, and as a comment in all changed or new source code files provided as part of your submission.*

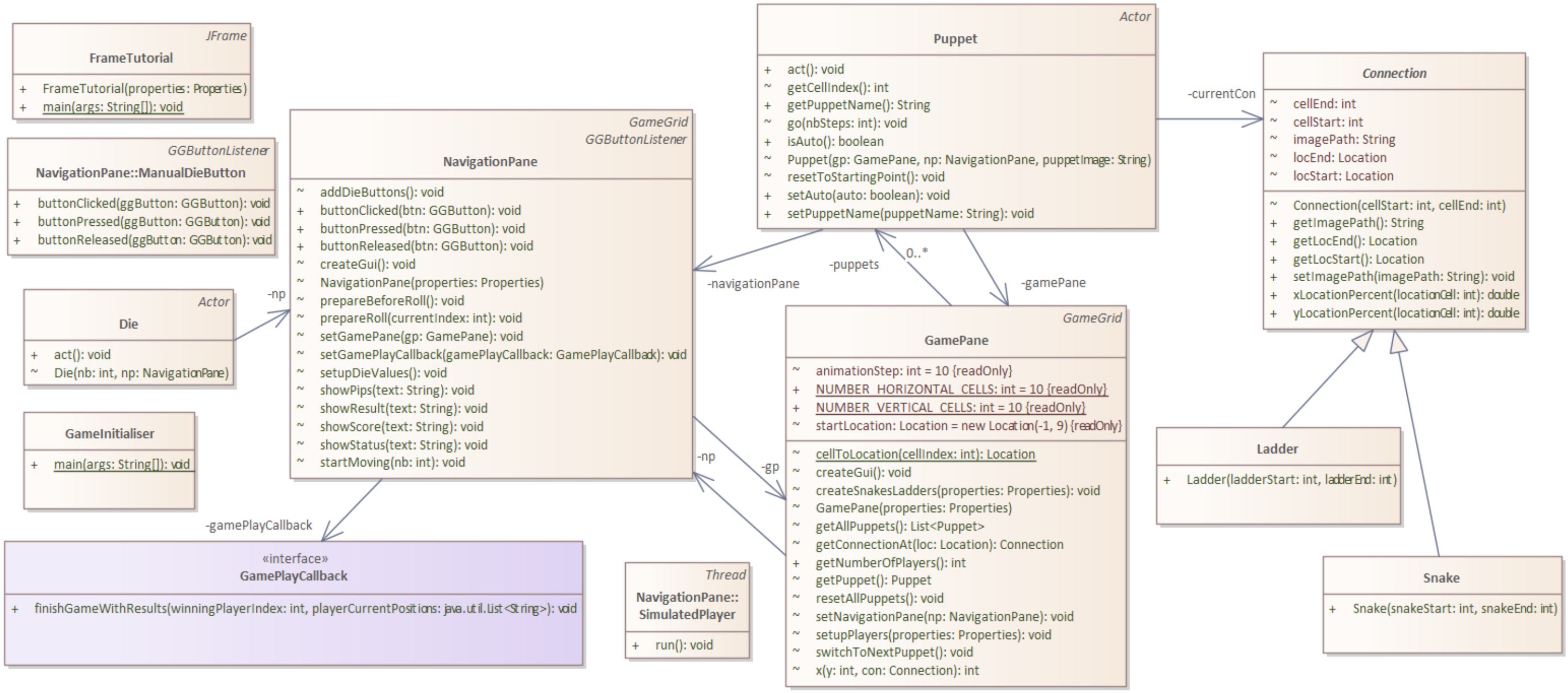

JFrame
FrameTutorial
+ FrameTutorial(properties: Properties)
+ main(args: String[]): void
GGButtonListener
NavigationPane::ManualDieButton
+ buttonClicked(ggButton: GGButton): void
+ buttonPressed(ggButton: GGButton): void
+ buttonReleased(ggButton: GGButton): void
Actor
Die
+ act(): void
~ Die(nb: int, np: NavigationPane)
GameInitialiser
+ main(args: String[]): void
-np
GameGrid
GGButtonListener
NavigationPane
~ addDieButtons(): void
+ buttonClicked(btn: GGButton): void
+ buttonPressed(btn: GGButton): void
+ buttonReleased(btn: GGButton): void
~ createGui(): void
~ NavigationPane(properties: Properties)
~ prepareBeforeRoll(): void
~ prepareRoll(currentIndex: int): void
~ setGamePane(gp: GamePane): void
~ setGamePlayCallback(gamePlayCallback: GamePlayCallback): void
~ setupDieValues(): void
~ showPips(text: String): void
~ showResult(text: String): void
~ showScore(text: String): void
~ showStatus(text: String): void
~ startMoving(nb: int): void
-gamePlayCallback
«interface»
GamePlayCallback
+ finishGameWithResults(winningPlayerIndex: int, playerCurrentPositions: java.util.List<String>): void
Thread
NavigationPane::
SimulatedPlayer
+ run(): void
-navigationPane
-np
-gp
Actor
Puppet
+ act(): void
~ getCellIndex(): int
+ getPuppetName(): String
~ go(nbSteps: int): void
+ isAuto(): boolean
~ Puppet(gp: GamePane, np: NavigationPane, puppetImage: String)
~ resetToStartingPoint(): void
+ setAuto(auto: boolean): void
+ setPuppetName(puppetName: String): void
-puppets
0..*
-gamePane
GameGrid
GamePane
~ animationStep: int = 10 {readOnly}
+ NUMBER_HORIZONTAL_CELLS: int = 10 {readOnly}
+ NUMBER_VERTICAL_CELLS: int = 10 {readOnly}
~ startLocation: Location = new Location(-1, 9) {readOnly}
~ cellToLocation(cellIndex: int): Location
~ createGui(): void
~ createSnakesLadders(properties: Properties): void
~ GamePane(properties: Properties)
~ getAllPuppets(): List<Puppet>
~ getConnectionAt(loc: Location): Connection
+ getNumberOfPlayers(): int
~ getPuppet(): Puppet
~ resetAllPuppets(): void
~ setNavigationPane(np: NavigationPane): void
~ setupPlayers(properties: Properties): void
~ switchToNextPuppet(): void
~ x(y: int, con: Connection): int
-currentCon
Connection
~ cellEnd: int
~ cellStart: int
~ imagePath: String
~ locEnd: Location
~ locStart: Location
~ Connection(cellStart: int, cellEnd: int)
+ getImagePath(): String
+ getLocEnd(): Location
+ getLocStart(): Location
+ setImagePath(imagePath: String): void
+ xLocationPercent(locationCell: int): double
+ yLocationPercent(locationCell: int): double
Ladder
+ Ladder(ladderStart: int, ladderEnd: int)
Snake
+ Snake(snakeStart: int, snakeEnd: int)

Figure 1 Partial Design Class Diagram for SLOP