s1950427

# Introduction to Practical Programming with Objects

# Assignment 2 - Sheepdog Trials Report

**Program Design**

The names of all packages and the names of all classes defined within them are defined below:

- main
    - SheepdogTrials
- animals
    - Animals
    - Dog
    - Sheep
- interfaces
    - InterfaceModel
    - InterfaceVew
- game
    - Model
    - FixedModel
    - RandomModel
    - TextView
    - Controller
- util
    - GameSettings
    - Util

Firstly, **SheepdogTrials** contained in the “main” package, is the main class of the project and creates a controller to start the game.

In the “animals” package, the **Animals** class can be found, which represents the objects that can be on the game board and focuses on moving objects, animals like the sheep and the dog. It is the parent class of the **Dog** and **Sheep** classes which represent the dog and the sheep of the game respectively.

Additionally, two interfaces were created in the “interfaces” package, named **InterfaceModel** and **InterfaceView,** where the methods defined in these interfaces, are those that must be implemented by a model and a view respectively. These interfaces were introduced to improve the level of abstraction and aid in a possible future extension of the program.

The implementation of the two interfaces happens in the “game” package, where the **Model** class implements **InterfaceModel** in order to represent the state of the game. This class is the parent class of **FixedModel** and **RandomModel** classes, which are used to represent the state of the game when the user asks for a fixed setup game and a random setup game respectively. Moreover, the **TextView** class implements **InterfaceView,** in order to interact with the user. In addition, in the “game” package there is the **Controller** class, which has as duty to control the flow of the game.

At last, in the “util” package, there is the **GameSettings** class, which contains the dimension of the board, the number of dogs, the number of sheep and the dimensions of the pen, and the **Util** class, which provides methods to be used by other classes, like a user input method and a method calculating the Euclidean distance between two points.

## Flow of execution

As described above, the flow of execution, is controlled by the **Controller** class, as the name suggests. When the main class runs, it creates a new controller and calls the **startSession()** method from the **Controller** class. This method displays a welcome message to the user and requests a menu selection. If the user enters '1', the **startNewGame()** method is called to start a fixed setup new game and if he enters '2', the **startRandomNewGame()** method is called to start a random setup new game. Both these methods initialise the corresponding game, by creating a new model using the **FixedModel** or the **RandomModel** classes, and then initialize the game.

In the case of a fixed setup game the board initialized has a fixed predefined setup, while for the case of a random setup game, various settings like board dimensions, pen dimensions, sheep number and their exact location are randomly generated to create the board. Note that the bounds of possible settings were carefully chosen so as the game is playable. Additionally, each animal's position and id are stored using the **Dog** and **Sheep** classes.

After the board of the game is initialized, the **startMatchLoop()** method in the controller is called, which contains a loop of instructions that are followed until the player wins or concedes. This method at the beginning displays the game state and the board to the user and begins the loop. In the beginning of the loop, the user is asked to enter a move and if he enters an invalid move, an error message explaining valid moves is displayed and he is asked for a valid move again and again until he enters one. After the user enters a valid move the chosen move is displayed, and the move is implemented using the model. In the case that the player didn't concede, the sheep behaviour is applied using the **model.sheepBehaviour()** method, which makes the sheep run away from the dog if they are sufficiently close to it or they start flocking, while being reluctant to enter the pen. Then the updated board is displayed, the counter of moves initialized before the loop is increased by one and the number of moves done so far is displayed to the user, to finish the loop. When the while loop finally breaks, the game state is displayed, using a victory or conceding message, and the user is offered to play again.

## Why this design is good

Firstly, using the interfaces, the level of abstraction of the program is increased, and there is clear division of tasks, i.e., only the **TextView** class can interact with the user and only the **Model** class can alter the game state. Also, the creation of the **Controller** class enables an easy access to the flow of execution of the game. Additionally, a hierarchy is created between classes, like **Model** and **FixedModel** or **RandomModel**, which helps to avoid duplication of code but also to inherit properties like methods from the parent class and still distinguish clearly between the two different models (set ups). At last, the **Dog** and **Sheep** classes may not be used extensively by the program but, like the use of the two interfaces, are a great tool for a potential extension of the game, like adding more dogs or personalize behaviour for each different sheep. I find the design very clear to follow, but also very easy to adjust or add anything in the game.