‘Selkies and Munros’ System Design and Specification

Introduction / Problem Statement

This project is a game of Selkies and Munros, based on the original game Snakes and Ladders. The goal of both games is a race to reach the 100th tile on a 10 by 10 2D board. Instead of snakes taking the players to lower on the grid tiles, these are replaced by selkies. Similarly with ladders taking players up tiles closer to reaching the 100th tile, Munros are used instead.
Unlike Snakes and ladders, these will have additional activity spaces like the Whiskey Boost, which will cause the player that lands on the tile to be moved 5 spaces ahead from that tile.

Additionally, the game will keep track of how many turns it tool the player to complete the game and display score board to the player in order of least turns

Requirements

Functional Requirements

R1. The system shall display a welcome menu with start, load, help options and leader board.
R.1.1 The system shall detect if the user’s selection.
R1.2 The system shall display the game’s instructions when the help option is selected.
R1.3 The system shall start a new game when the start option is selected.
R1.4 The system shall load a game the user selects.
R1.5 When the load option is selected, the user shall be given a list of save files to select.
R1.6 When the leader board option is selected the game shall display the top score to the player

R2. Once the user has selected start, the system shall ask the user how many players there are.
R2.1 The system should limit the game’s players to the amount that is received from the user.
R2.2 The system should validate that the number of players is between 1 and 4.

R3. The system shall prompt users to enter a username.
R3.1 The system should validate the username entered to ensure they are no longer than 10 characters and no less than 3.
R3.2 The system should address users to the username they have chosen.

R4. The system shall display a 10 by 10 2D grid.
R4.1 The grid shall be made of 100 square tiles.
R4.2 The grid shall number these tiles from 1 to 100.

R5. The grid shall contain normal tiles and obstacle tiles.
R5.1 Selkies and Munros shall be obstacle tiles.
R5.2 Obstacle tiles shall not move from their tiles.

R6. The grid shall contain activity tiles.
R.6.1 The activity tiles shall be generated on the grid at random.
R.6.2 The activity tile shall not be known to the user until they have landed on it.
R6.3 Once landed, the user shall be notified they’ve landed on it.

R.6.4 The activity tiles shall contain an activity called 'Whisky Boost' or 'Haggis Bite' or 'Nessie's Revenge' or 'Bagpiper's Delight'.
R.6.5 If the player lands on a Whiskey boost tile they shall move an additional 5 spaces.
R.6.6 If the player lands on a Haggis Bite tile, they shall lose a turn.
R.6.7 If the player lands on a Nessie's Revenge tile, all players will move to a random position.
R.6.8 If the player lands on a Bagpiper's Delight tile, they will roll again.
R.6.9 if the player lands on another activity tile after the activity tile's effects or obstacle tile their rules shall still apply. Unless the player has landed on Nessie's Revenge.
R6.10 An activity tile's position shall not be the same as another activity tile or obstacle tile position.

R7. Obstacle tiles shall have a pointer.
R7.1 The pointer shall be a random position on the board for the player to move to.
R7.2 If the obstacle tile is a Munro the pointer shall only send the player to a position above the obstacle.
R7.3 if the obstacle is a Selkie the pointer shall only send the player to a position below the obstacle.
R7.4 If the player is on a obstacle they shall be moved to the pointer's position.
R7.5 The player shall not move from a pointer to the position of the obstacle.

R8. Each round shall allow each player to roll a 6-sided dice.
R8.1 Each round shall not end until each player has taken a turn rolling a dice.
R8.2 Players shall take one turn rolling a dice.
R8.3 Players shall take turns one after the other.

R9. Each turn the player shall be given the option save the game.
R9.1 The player shall be given their turn back once the game is saved.

R10. The system shall display which player's turn it is during the round.
R10.1 The system should use the player's chosen username when displaying the player's turn.

R11. The player shall move tiles according to the number they rolled on their dice.
R11.1 The player shall not move back any tiles, when using the dice.

R.12 Once a player has reached the 100th tile the game shall set that player as the winner.
R.12.1 The game shall display a message notifying the player they have won
R12.2 The game shall update the leader board with how many tuns it took the player to win as well as their username.
R12.3 the game shall return to the main menu

R.13 All players shall start on tile 1 (
R.13.1 Once they've rolled the dice in their turn, the outcome shall be added to their current location

Non-functional Requirements

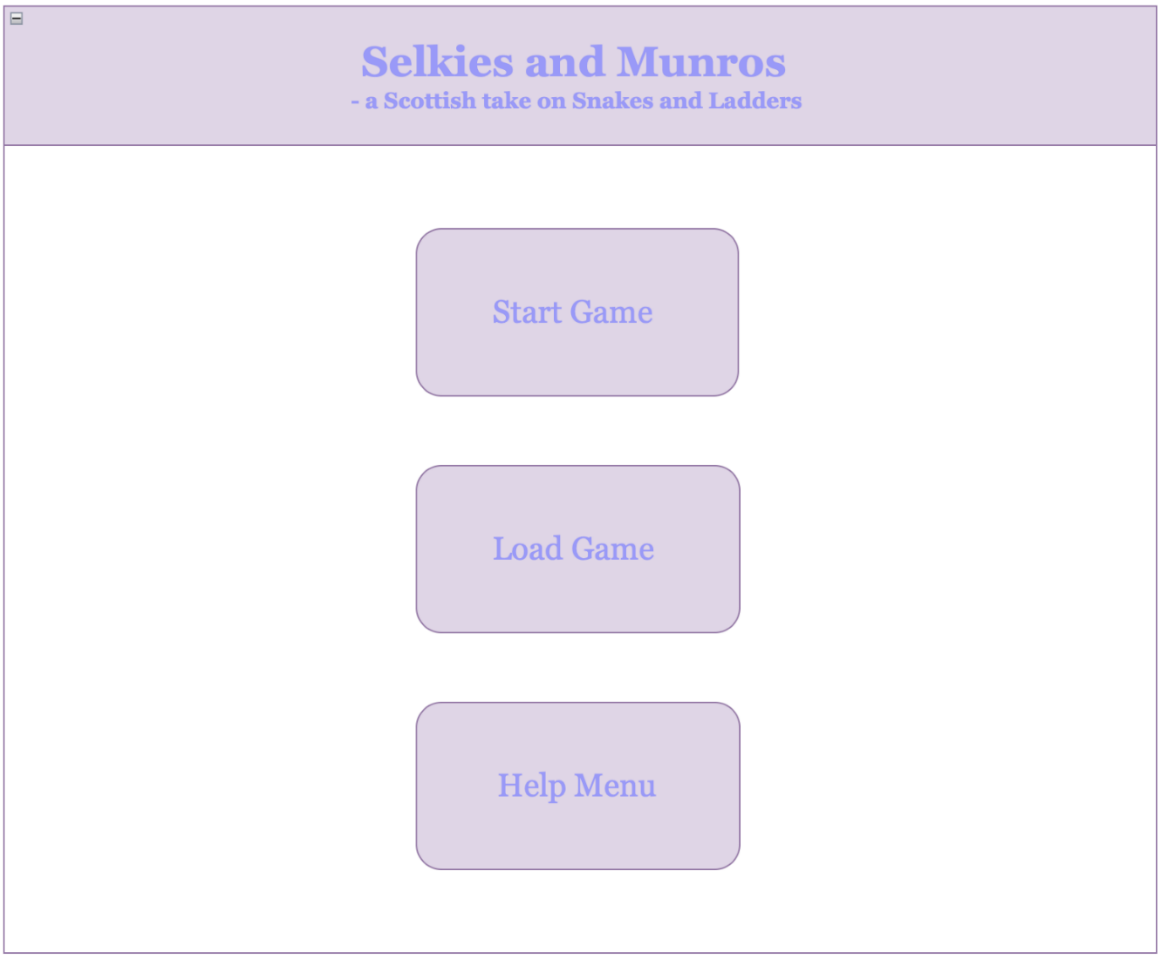
NFR1. The program shall be written in Java.

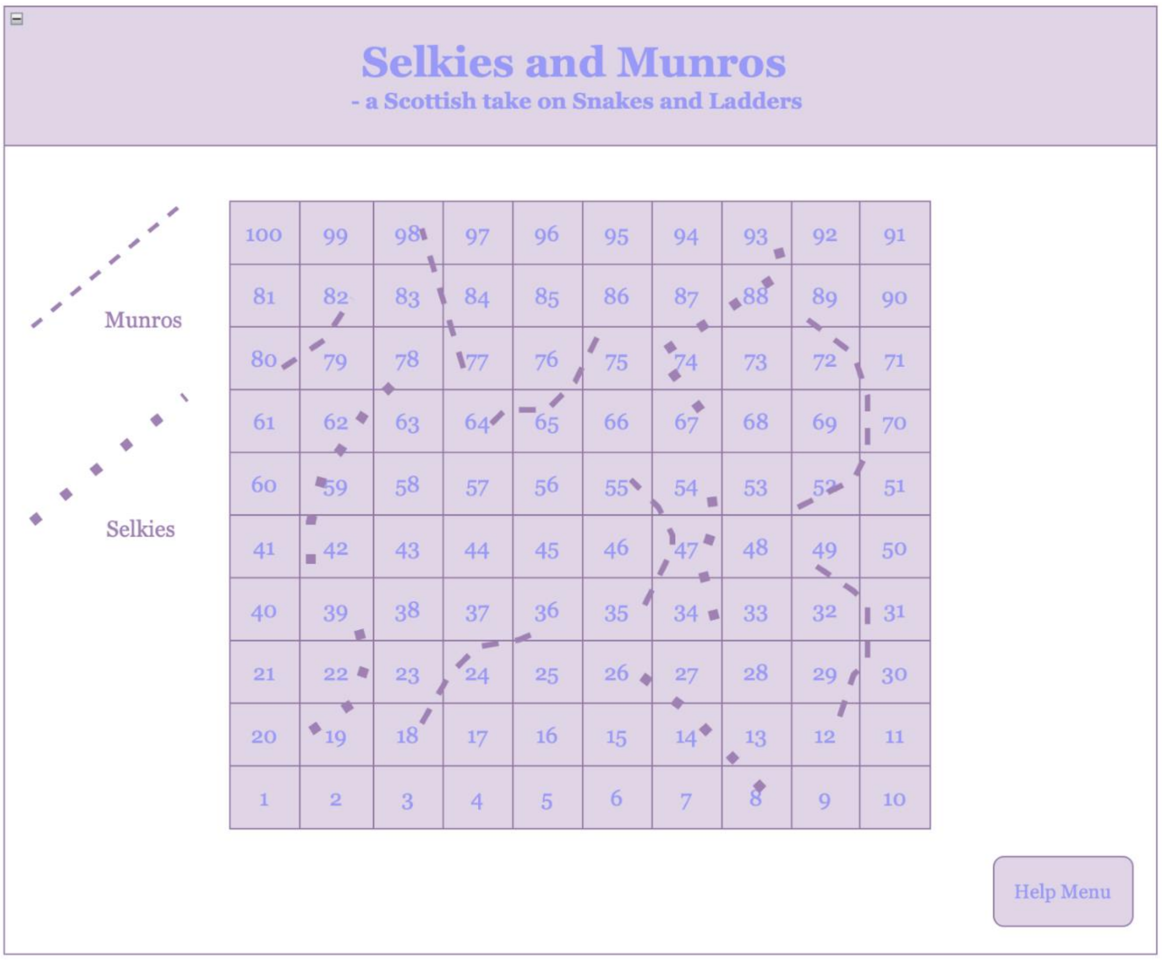
NFR2. The system shall be completed no later than 11.59pm on Sunday 3rd December 2023.

NFR3. The system should allow 1-4 player to play at once.

NFR4. The system shall save the player's game information on an external file.

User Interface

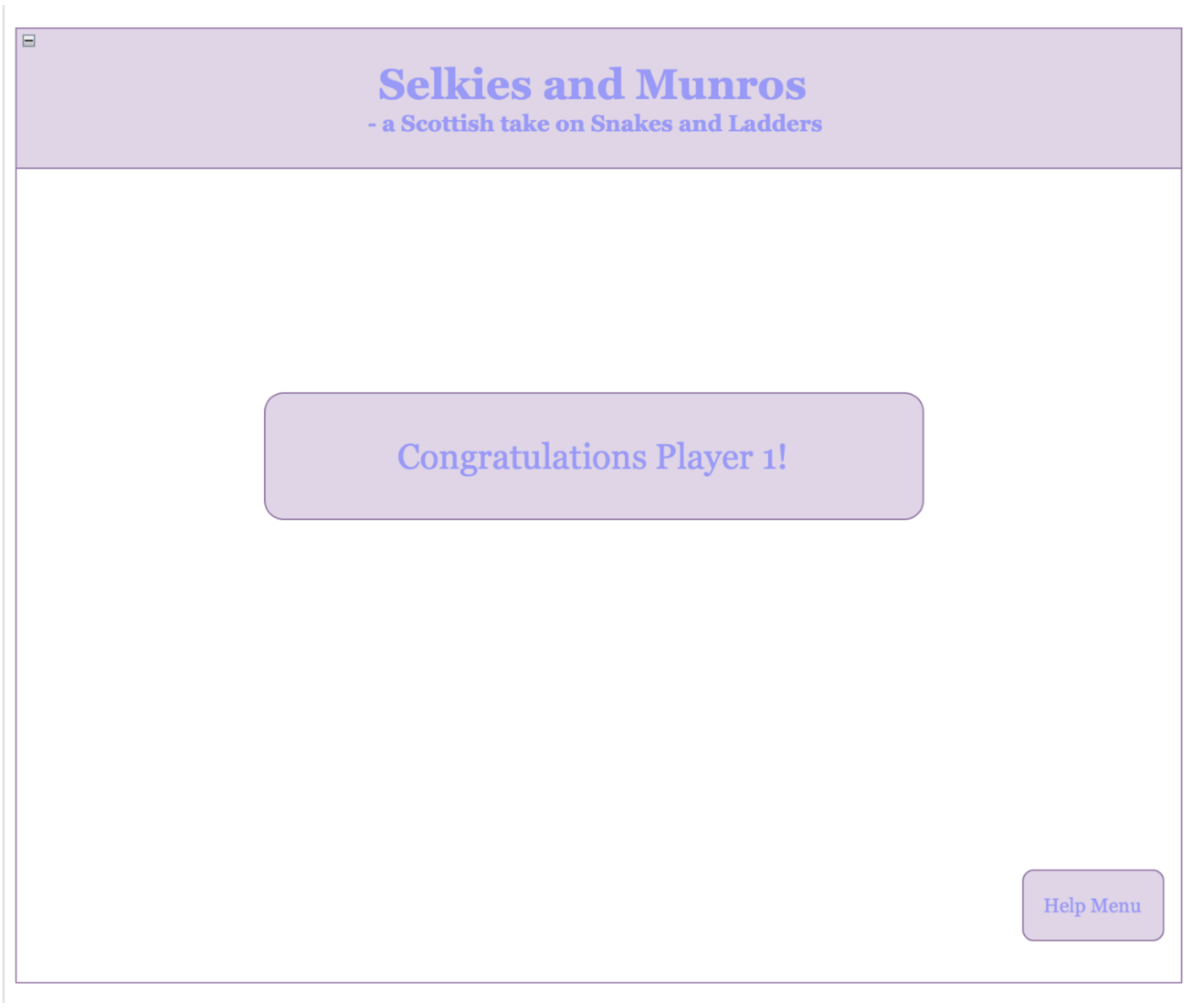




Selkies and Munros

- a Scottish take on Snakes and Ladders

100	99	98	97	96	95	94	93	92	91
81	82	83	84	85	86	87	88	89	90
80	79	78	77	76	75	74	73	72	71
61	62	63	64	65	66	67	68	69	70
60	59	58	57	56	55	54	53	52	51
41	42	43	44	45	46	47	48	49	50
40	39	38	37	36	35	34	33	32	31
21	22	23	24	25	26	27	28	29	30
20	19	18	17	16	15	14	13	12	11
1	2	3	4	5	6	7	8	9	10


Selkies and Munros
- a Scottish take on Snakes and Ladders
Congratulations Player 1!
Help Menu

Use Cases

Play Selkies and Munros			Alternatives
1	User:	Starts programme	
2	System:	Displays main menu page, containing a welcome message, an option to start the game and load , help options and a leaderboard	A1, A2
3	User:	Selects starts game	
4	System:	Starts a new game	
5	System:	Asks user how many players there are	
6	User:	Enters how many players there are	
7	System:	Confirms that it is between 1-4 players	
8	System:	Prompts users to enter a username	
9	System:	Confirms that username is greater than 3 but no longer than 10 characters	
10	System:	Addresses users by the given usernames	
11	System:	Displays Tile grid of Selkies and Munros to user	
12	System:	Prompts user to roll dice	
13	User:	Rolls dice	
14	System:	Moves player on grid according to the number given by the dice	A3
15	System:	Displays that it is the next player turn	

		Steps 12-15 repeated for all players, until one player reaches the 100th tile	A4, A5
16	System:	Displays win message to player	
17	System:	returns to main menu	
18	User:	Selects from options, or exits game	

A1 – Load Game			Alternatives
1	User:	Selects Loads game	
2	System:	Displays a list of save files	
3	User:	Selects a save	
4.	System:	Loads save file	Return to 10 in Play Selkies and Munros

A2– Help menu			Alternatives
1	User:	Selects help menu	
2	System:	Displays a description with instruction on how to play the game	
3.	System:	Gives the option to return to the main menu	Return to 2 - Selkies and Munros

A3- Special Tile			Alternatives
1	System:	User has landed on whiskey boost; system displays message stating this	
2	System:	Moves player an additional 5 tiles	
3	System:	Proceeds with 12-15 – Selkies and Munros	

A4 – User Saves game			Alternatives
1	User:	Selects save game option	
2	System:	Prompts the user to enter a name for the save file	
3.	User:	Enters file name	
4.	System	Saves game info	Return to 12 - Selkies and Munros

A5 – User returns to the main menu			Alternatives
1	User:	Selects the option to return to main menu	
2	System:	Returns to 2 - Selkies and Munros	

Classes

Classes
Player

Tiles
Obstacles(Child of Tiles)
FileAction
GenerateRandom
Game
Menu
Board

Class Descriptions including Responsibilities, Fields, and Methods

Player:
Fields - position(int), username(string), turn(boolean), image(string[ASCII])
Methods – getUsername (asks for player to input their username and stores it), getTurn (True or false if it's the player's turn or not), updatePos (updates the position of the player on the board), move(Decides which way the player should move and based of the a int input will move them the correct direction and spaces)
Role: Allows a visual representation of where the player is, stores and allows a display of their chosen username.

Tile:
Fields – type(String), image(String), defaultImage(String)
Methods – setImage(sets the image of the tile), getImage(returns the image of the tile), getDefaultImage(Return the default image of the tile), getType(Returns what type of tile it is e.g Tile or Munro), reset(Resets the image to the initial default image)
Role: Each tile makes up the grid in the game class. Helps with game logic for each index in the grid.

Obstacle(Child of Tile):
Fields – pointer(Array of integers)
Methods – getpointer(Returns the pointer field)
Role: Needed because both Selkies and Munros need a pointer for the player to go to once they have landed on them.

FileAction:
Methods: save(writes an objects data to a file), load(loads an objects data from a file), checkDir(checks if a directory exists and if not creates a new one) checkFile(Checks if a file exists and if not creates a new one), writeToFile(writes to a file, creating a new line), readFile(reads a given file and returns it's contents)
Role: To make it easier to mange reading and writing from files, without having to create multiple methods in different classes.

GenerateRandom:
Methods: genRandomInt(generates a random integer value)
Role: To handle the generation of random numbers

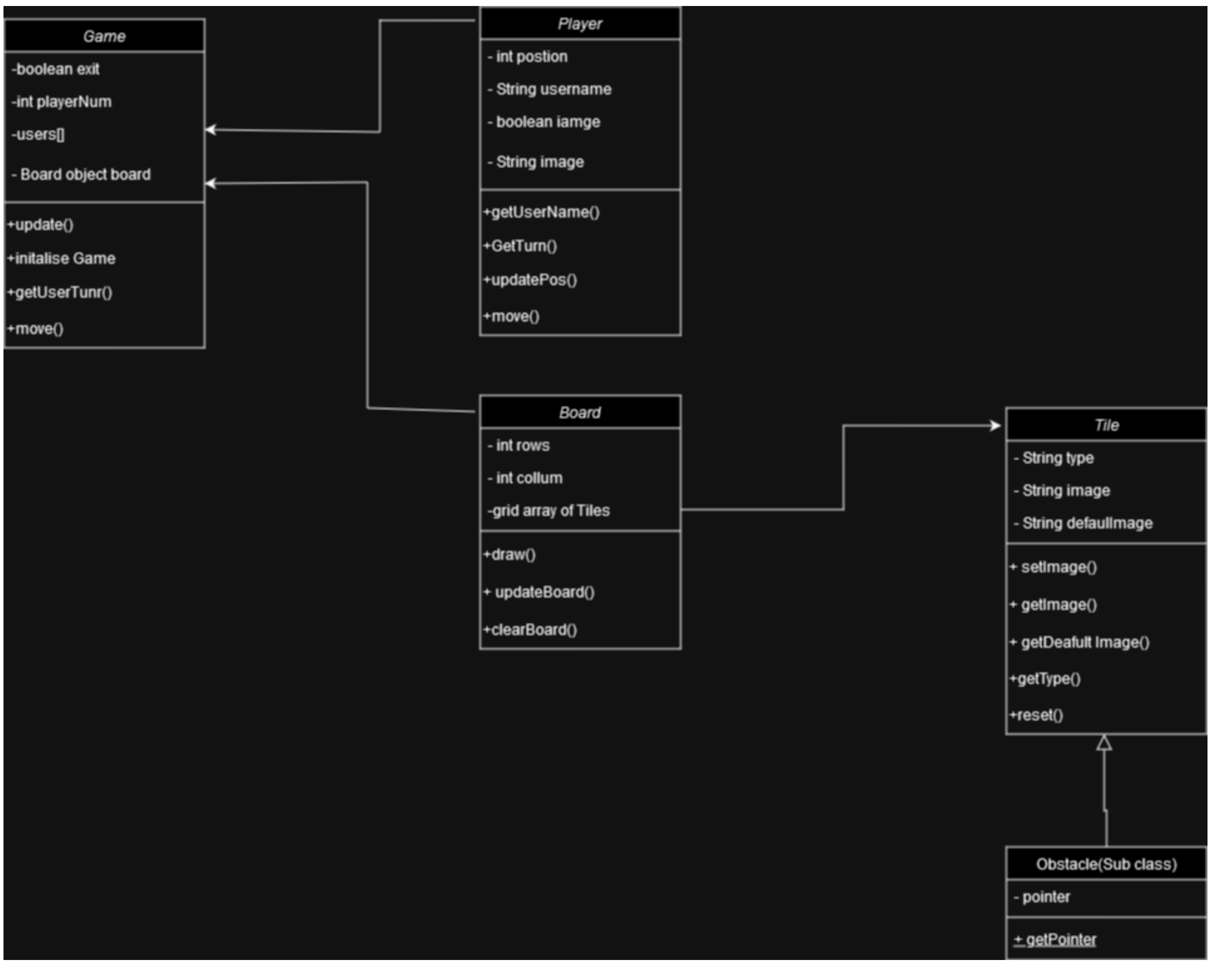
Menu:
Fields – exit(boolean), game(a Game object), leaderboard(An 2D array containg the leader boards values)
Methods – runMenu(Runs the main menu of the program until exit is true), drawMenu(Displays the menu to the screen), action(Executes an action based of the user's choice e.g. start game), runGame(Runs the main game), gameAction(Depending on user's choice executes an action e.g roll

dice or save game), saveGame(save the current game), loadGame(Loads a game from a save file), displayLeaderBoard(Displays the leader board to the user), displayHelp(displays the help/instuctions)
Role: Displays an interactive menu to the user's and executes actions based on their choice. Also uses the logic from a game object to run the game.

Board:
Fields – rows(int), collums(int) grid(Array of Tile objects)
Methods: draw(draws the grid to the screen), updateBoard(updates the position and image of an object in the board), clearBoard(puts the board back to its default state)
Role: A visual representation of their all the objects are on the screen

Game:
Fields - exit(boolean), playerNumber(int, number of players), users(Array of Player Objects), board(An instance of a Borad Object), turn(int).
Methods – update(updates the screen with player movement), intialiseGame(Gets the number of players before the game starts), getUserTurn(Returns if it's the current user's turn or not), rollDice(Rolls a 6 sided dice and move the player accordingly), checkType(Checks what tile type theplayer has landed on moves them depending on the type). checkWin(Checks if the user has won or not), updateLeaderBoard(Updates the leader board with the new winner's information)

Class Diagram



Activity Diagrams / Pseudocode

```
Class Player
Move() method input(integer diceRoll)
    Set integer temp to equal to current x position
    Set integer diff to equal to 0
    If current y position is odd then
        Player moves left
        If current x position < 0 then
            Player moves up the grid
            diff = temp
            player x position = (diceRoll - diff) -1
    Else
        Player moves right
        If current x position is > 9 then
            Player moves up the grid
            Diff = the difference between 9 and temp
            Player x position = 9 – (diceRoll - diff) + 1
End

Class Game
CheckType() method input(integer user, integer array userPos)
    Set String type to the gridOfTiles[userPos] type attrribute
        If type is != “Tile” then
            If type is equal to Munro or Selkie then
                Set ArrayOfPlayers[user] position to gridOfTiles[userPos] pointer
                Update ArrayOfPlayers[user] on board
            Else if type = “HaggisBite” then
                ArrayOfPlayers[user] skip turn
                Update ArrayOfPlayers[user] on board
            Else if type = “NessieRevenge” then
                For i = 0 to length of ArrayOfPlayer loop
                    ArrayOfPlayer[i] postion set to random
                    Update ArrayOfPlayers[i] on board
            Else if type = “BagpiperDelight” then
                ArrayOfPlayers[user] rolls the dice again
                Update ArrayOfPlayers[user] postion on board
            Else if type = “WhiskyBoost” then
                ArrayOfPlayers[user].move(5)
                checkType(user, userPos)
                Update ArrayOfPlayers[user] postion on board
        Else
            Update ArrayOfPlayers[user] postion on board
End
```