Assignment 04

Group 02

Exercise 1 - Code improvements

1. This week, we replaced **six scenarios with switch/case statements** and **four scenarios with if/else statements**.

Switch/case statements

1. AreaTracker.printBoardGrid()

```
+       Map<AreaState, String> areaStateVisualisation = new HashMap<AreaState,
String>();
+       areaStateVisualisation.put(AreaState.OUTERBORDER, "[X]");
+       areaStateVisualisation.put(AreaState.INNERBORDER, "[*]");
+       areaStateVisualisation.put(AreaState.UNCOVERED, "[ ]");
+       areaStateVisualisation.put(AreaState.FAST, "[F]");
+       areaStateVisualisation.put(AreaState.SLOW, "[S]");
        for (AreaState[] column : boardGrid) {
          for (AreaState state : column) {
-             switch (state) {
-                case OUTERBORDER:
-                   System.out.print("[X]");
-                   break;
-                case INNERBORDER:
-                   System.out.print("[*]");
-                   break;
-                case UNCOVERED:
-                   System.out.print("[ ]");
-                   break;
-                case FAST:
-                   System.out.print("[F]");
-                   break;
-                case SLOW:
-                   System.out.print("[S]");
```

```
-                 break;
-             default:
-                 break;
-         }
```

Implemented a map describing the relations between AreaStates and their log output string.

2. AreaTracker.permuteLocation()

```
-     switch (quadrant) {
-         case 1:
-             x += threadLocalRandom.nextInt(-1, 1);
-             y += threadLocalRandom.nextInt(-1, 1);
-             break;
-         case 2:
-             x += threadLocalRandom.nextInt(0, 2);
-             y += threadLocalRandom.nextInt(-1, 1);
-             break;
-         case 3:
-             x += threadLocalRandom.nextInt(-1, 1);
-             y += threadLocalRandom.nextInt(0, 2);
-             break;
-         case 4:
-             x += threadLocalRandom.nextInt(0, 2);
-             y += threadLocalRandom.nextInt(0, 2);
-             break;
-         default:
-             x += threadLocalRandom.nextInt(-1, 2);
-             y += threadLocalRandom.nextInt(-1, 2);
-             break;
+
+     Map<Integer, List<Integer>> locationMap = new HashMap<>();
+     locationMap.put(1, Arrays.asList(-1, 1, -1, 1));
+     locationMap.put(2, Arrays.asList(0, 2, -1, 1));
+     locationMap.put(3, Arrays.asList(-1, 1, 0, 2));
+     locationMap.put(4, Arrays.asList(0, 2, 0, 2));
```

```
+       List<Integer> locations = locationMap.get(quadrant);
+       if (locations != null) {
+           x += threadLocalRandom.nextInt(locations.get(0), locations.get(1));
+           y += threadLocalRandom.nextInt(locations.get(2), locations.get(3));
+       } else {
+           x += threadLocalRandom.nextInt(-1, 2);
+           y += threadLocalRandom.nextInt(-1, 2);
```

Implemented a locationMap containing the information about the locations that should be changed according to the quadrant value.

3. AreaTracker.getCornerCoordinates()

```
-       switch (quadrant) {
-           case 1:
-               x = 0;
-               y = 0;
-               break;
-           case 2:
-               x = Globals.BOARD_WIDTH / 2;
-               y = 0;
-               break;
-           case 3:
-               x = 0;
-               y = Globals.BOARD_HEIGHT / 2;
-               break;
-           case 4:
-               x = Globals.BOARD_WIDTH / 2;
-               y = Globals.BOARD_HEIGHT / 2;
-               break;
-           default:
-               x = 0;
-               y = 0;
-               break;
+       Map<Integer, List<Integer>> quadrantCornerMap = new HashMap<>();
+       quadrantCornerMap.put(1, Arrays.asList(0, 0));
+       quadrantCornerMap.put(2, Arrays.asList(Globals.BOARD_WIDTH / 2, 0));
```

```
+       quadrantCornerMap.put(3, Arrays.asList(0, Globals.BOARD_HEIGHT / 2));
+       quadrantCornerMap.put(4, Arrays.asList(Globals.BOARD_WIDTH / 2,
Globals.BOARD_HEIGHT / 2));
+       List<Integer> integers = quadrantCornerMap.get(quadrant);
+       if (integers != null) {
+           x = integers.get(0);
+           y = integers.get(1);
+       } else {
+           x = 0;
+           y = 0;
```

Implemented a quadrantCornerMap which contains the relationships between quadrant values and the corresponding x and y values.

4. GameController.spawnPowerup()

```
-           switch (PowerUpType.randomType()) {
-               case EAT:
-                   powerup = new PowerEat(coordinates[0], coordinates[1],
-                           Globals.BOARD_MARGIN * 2, Globals.BOARD_MARGIN * 2,
areaTracker);
-                   break;
-               case LIFE:
-                   powerup = new PowerLife(coordinates[0], coordinates[1],
-                           Globals.BOARD_MARGIN * 2, Globals.BOARD_MARGIN * 2,
areaTracker);
-                   break;
-               case SPEED:
-                   powerup = new PowerSpeed(coordinates[0], coordinates[1],
-                           Globals.BOARD_MARGIN * 2, Globals.BOARD_MARGIN * 2,
areaTracker);
-                   break;
-               case NONE:
-                   return;
+           Map<PowerUpType, Powerup> powerupMap = new HashMap<>();
+           powerupMap.put(PowerUpType.EAT, new PowerEat(coordinates[0],
coordinates[1],
+                   Globals.BOARD_MARGIN * 2, Globals.BOARD_MARGIN * 2,
areaTracker));
```

```
+            powerupMap.put(PowerUpType.LIFE, new PowerLife(coordinates[0],
coordinates[1],
+                    Globals.BOARD_MARGIN * 2, Globals.BOARD_MARGIN * 2,
areaTracker));
+            powerupMap.put(PowerUpType.SPEED, new
PowerSpeed(coordinates[0], coordinates[1],
+                    Globals.BOARD_MARGIN * 2, Globals.BOARD_MARGIN * 2,
areaTracker));
+            powerup = powerupMap.get(PowerUpType.randomType());
+            if (powerup == null) {
+                return;
```

Instead of making a Powerup when some PowerUpType is met, create initialize the different Powerups beforehand and pick one from a map containing relationships between PowerUpTypes en Powerups.

5. PowerUpType.randomType()

```
-       switch (rand) {
-           case 0:
-               return EAT;
-           case 1:
-               return LIFE;
-           case 2:
-               return SPEED;
-           default:
-               return NONE;
+       Map<Integer, PowerUpType> powerUpTypeMap = new HashMap<>();
+       powerUpTypeMap.put(0, EAT);
+       powerUpTypeMap.put(1, LIFE);
+       powerUpTypeMap.put(2, SPEED);
+       PowerUpType powerUpType = powerUpTypeMap.get(rand);
+       if (powerUpType == null) {
+           return NONE;
        }
+       return powerUpType;
```

This couples integer values (a random number picked between 0 and 2) with a PowerUpType using a powerUpTypeMap.

6. Cursor.move()

```
-            if (currentMove.equals(arrowKeys.get(2))) {
```

```
-                transX = -1;
-            } else if (currentMove.equals(arrowKeys.get(3))) {
-                transX = 1;
-            } else if (currentMove.equals(arrowKeys.get(0))) {
-                transY = -1;
-            } else if (currentMove.equals(arrowKeys.get(1))) {
-                transY = 1;
-            }
+            // A map containing relationships between keycodes and the movement directions.
+            Map<KeyCode, CursorMovement> cursorMovementMap = new HashMap<>();
+            cursorMovementMap.put(arrowKeys.get(2), new CursorMovement(-1, 0));
+            cursorMovementMap.put(arrowKeys.get(3), new CursorMovement(1, 0));
+            cursorMovementMap.put(arrowKeys.get(0), new CursorMovement(0, -1));
+            cursorMovementMap.put(arrowKeys.get(1), new CursorMovement(0, 1));
+            transX += cursorMovementMap.get(currentMove).getTransX();
+            transY += cursorMovementMap.get(currentMove).getTransY();
```

Added an extra class, namely CursorMovement which contains integers transY and transX. The CursorMovement class describes in which direction the cursor should move next. Instead of writing an if/else statement checking all arrowKeys, we created a cursorMovementMap which describes the relations with a CursorMovement and its corresponding arrow keys.

7. SparxDirection.randomDirection()

```
-    switch (rand) {
-        case 0:
-            return LEFT;
-        case 1:
-            return RIGHT;
+    Map<Integer, SparxDirection> sparxDirectionMap = new HashMap<>();
+    sparxDirectionMap.put(0, LEFT);
+    sparxDirectionMap.put(1, RIGHT);
+    SparxDirection sparxDirection = sparxDirectionMap.get(rand);
```

```
+       if (sparxDirection == null) {
+           return LEFT;
        }
-       return LEFT;
+       return sparxDirection;
```

This methods returns a random direction, namely LEFT or RIGHT. rand is a random integer 0 or 1. We replaced the switch/case statement with a map describing the relations between the random integer and its corresponding SparxDirection.

8. AreaTrackerTest.testConstructor()

```
        for (int x = 0; x < expectedGrid.length; x++) {
            for (int y = 0; y < expectedGrid[x].length; y++) {
-               if (x == 0) {
-                   expectedGrid[x][y] = AreaState.OUTERBORDER;
-               } else if (x == expectedGrid.length - 1) {
-                   expectedGrid[x][y] = AreaState.OUTERBORDER;
-               } else if (y == 0 || y == expectedGrid[x].length - 1) {
-                   expectedGrid[x][y] = AreaState.OUTERBORDER;
-               } else {
-                   expectedGrid[x][y] = AreaState.UNCOVERED;
-               }
+               expectedGrid[x][y] = AreaState.UNCOVERED;
            }
        }
+
+       if (expectedGrid.length > 0) {
+           for (int i = 0; i < expectedGrid.length; i++) {
+               expectedGrid[i][0] = AreaState.OUTERBORDER;
+               expectedGrid[i][expectedGrid[0].length - 1] =
AreaState.OUTERBORDER;
+           }
+
+           for (int i = 0; i < expectedGrid[0].length; i++) {
+               expectedGrid[0][i] = AreaState.OUTERBORDER;
+               expectedGrid[expectedGrid[0].length - 1][i] =
AreaState.OUTERBORDER;
```

```
+         }
+     }
```

This method sets all the outerborders on the first and last values of x and on the first and last values of y. Instead of writing 4 if/else statements within the loop, set all states to innerborders beforehand and then set all outerborders by using two extra for loops.

9. AreaTracker constructor

```
-             //If the current row is the first row set all grid points border on that row
-             if (j == 0) {
-                 boardGrid[j][i] = AreaState.OUTERBORDER;
-             }
+              //By default, all points are uncovered
+              boardGrid[j][i] = AreaState.UNCOVERED;
+         }
+     }
+
+     for (int i = 0; i < boardGrid.length; i++) {
+         //If current column is the last column set the grid point on that column and the current row border
+         boardGrid[0][i] = AreaState.OUTERBORDER;
+         boardGrid[boardGrid[0].length - 1][i] = AreaState.OUTERBORDER;
+     }
+
+     if (boardGrid.length > 0) {
+         for (int j = 0; j < boardGrid[0].length; j++) {
              //If the current row is the bottom row set all grid points border on that row
-             else if (j == boardGrid[i].length - 1) {
-                 boardGrid[j][i] = AreaState.OUTERBORDER;
-             }
-             //If current column is the last column set the grid point on that column and the current row border
-             else {
-                 if (i == 0) {
-                     boardGrid[j][i] = AreaState.OUTERBORDER;
-                 }
```

```
-                //If the current column is the last column,
-                // set the grid point on that column and the current row border
-                else if (i == boardGrid.length - 1) {
-                    boardGrid[j][i] = AreaState.OUTERBORDER;
-                }
-                //If the current point is none of the above set that point to uncovered
-                else {
-                    boardGrid[j][i] = AreaState.UNCOVERED;
-                }
-            }
+            boardGrid[j][0] = AreaState.OUTERBORDER;
+            boardGrid[j][boardGrid.length - 1] = AreaState.OUTERBORDER;
```

Just like #8, rewrites sets the boardGrid AreaStates using 3 for-loops instead of one loop with 4 if-statements.

10. oppositeQuadrant() in cursor.java

At first the quadrants were arranged in the following order:

1	2
3	4

By replacing that order with this one:

0	1
3	2

Instead of using a switch statement in the oppositeQuadrant method the following formula could be use: oppositeQuadrant = (currentQuadrant + 2) % 4.

```
184  186      public int oppositeQuadrant() {
185  187          int quadrant = this.quadrant();
186  188
187       -      switch (quadrant) {
188       -          case 1:
189       -              return 4;
190       -          case 2:
191       -              return 3;
192       -          case 3:
193       -              return 2;
194       -          case 4:
195       -              return 1;
196       -          default:
197       -              return 1;
198       -      }
     189  +      return (quadrant + 2) % 4;
199  190      }
```

Exercise 2 - Teaming up

1. The requirements document for the new game feature can be found in **docs/Sprint #4/Own Improvement/Requirements Document Powerup**.
2. The UML diagram can be found in **docs/diagrams/PowerUpDiagram.png**

Exercise 3 - Walking in your TA’s shoes

Feedback for the other codebase can be found in **docs/Sprint #4/Evaluating code of others**