CSE443 - Object-Orianted Analysis & Desing - HW 2

Fatih Kaan Salgır - 171044009

Design Explanation

Java FX Tile Game

`Main` class creates the factories, loads the FXML template and shows the stage. `MainController` implements `Initializable` to initialize the Grid Pane and the Hero Pane. For displaying tiles, `GridPanel` is used. Every tile on the grid has a `Label`. Event handlers are added to these labels in order to listen the user input.

Hero pane consist of labels and a progress bar placed in a vertical box. Statusbar consist of a JavaFX's `ListView`.

`common/Util.java` stores the constants used different classes like the width & height of the window, animation durations etc.

Styling

Most of the styling of the FX elements made using CSS stylesheet (`app.css`). When the style needed to change, such as selecting a tile or highlighting a tile, classnames are added to the `Node` or removed from the `Node`.

Notes

- User input is disabled when the animation are running.
- The turn stays in the user, until he/she makes a successful move. (3 tiles lined up).
- Computer is waiting for 2 seconds before it makes a move.

Synchronization & Animations

When user selects 2 tiles, and the tiles are swapped, `cascadedMove` function calls move function recursively until the grid is cleared from lined up tiles. `move` takes another `Thread` as an arguement to wait, to enable cascading `move` calls. `waitAndRun` function first waits for the given thread to terminate by calling `Thread.join()`, runs the runnable and, sleeps for given miliseconds.

Every step of a move runs in a seperate thread to make the steps wait for each other before running.

Steps:

- wait for swap
- highlight matches
- remove matches
- fill gaps
- spawn new tiles

According to JavaFX the user interface cannot be directly updated from a non-application thread. Therefore UI updates made by the seperate thread wrapped up with `Platform.runLater(() -> {})`.

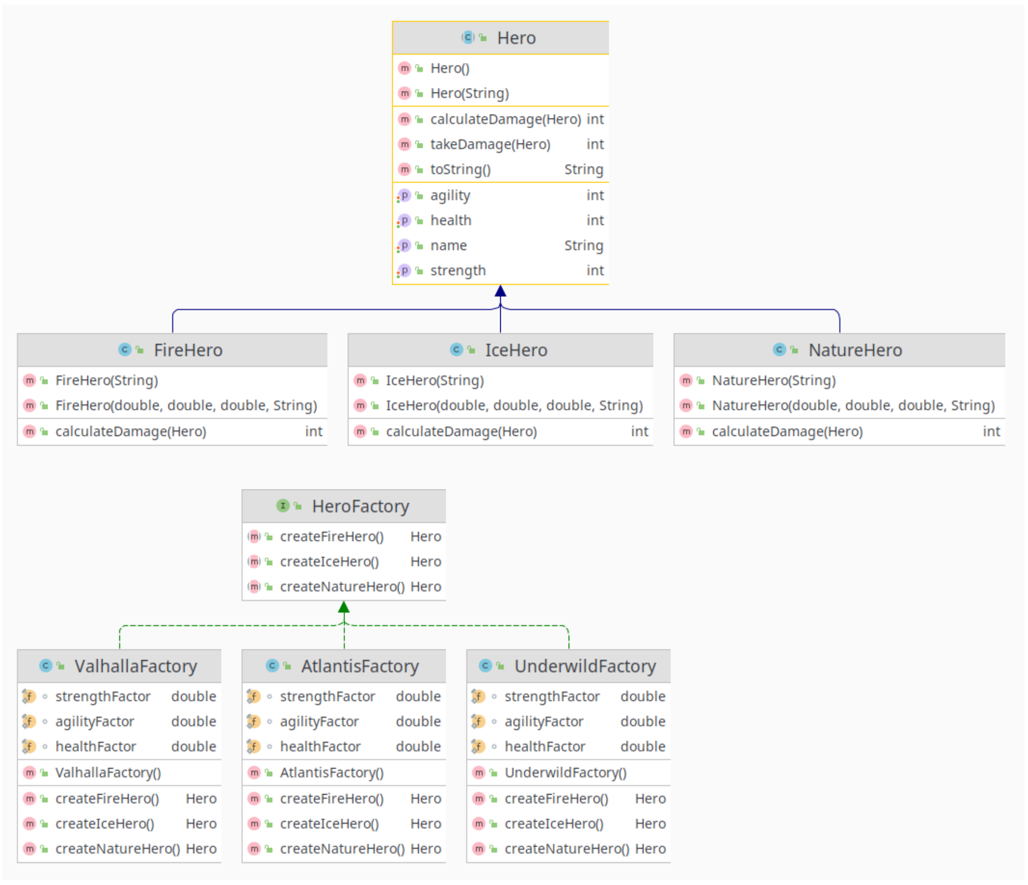
`move` function in the `Grid.java`;

```java
public Thread move(Thread waitFor, boolean isSwap) {
  boolean noMatch = !matchingTiles(t -> true);
  Thread swapWait = waitAndRun(null, () -> {
  }, Util.SWAP_DURATION);
  Thread highlight = waitAndRun(isSwap ? swapWait : null, () -> {
    highlighted = matchingTiles(TileView::highlight);
  }, noMatch ? 0 : Util.HIGHLIGHT_DURATION);

  Thread scaleDown = waitAndRun(highlight, () -> {
    // remove status
    Platform.runLater(() -> statusBar.getItems().clear());
    if (highlighted) {
      performDamage(characters, enemies);
      removeHighlight();
      removeMatchingTiles();
      // update ui
      Platform.runLater(() -> enemies.asList().forEach(HeroView::updateView));
    }
  }, noMatch ? 0 : Util.SCALE_DURATION);

  Thread fillGaps = waitAndRun(scaleDown, () -> {
    fillGaps();
  }, noMatch ? 0 : Util.FILL_GAPS_DURATION);

  Thread spawnNewTiles = waitAndRun(fillGaps, () -> {
    spawnNewTiles();
    if (highlighted) {
      changeTurn();
    } else {
      userInputEnabled = true;
    }
  }, noMatch ? 0 : Util.FILL_GAPS_DURATION);

  waitAndRun(waitFor, () -> {
    swapWait.start();
    highlight.start();
    scaleDown.start();
    fillGaps.start();
    spawnNewTiles.start();
  }, 0).start();
  return spawnNewTiles;
}
```

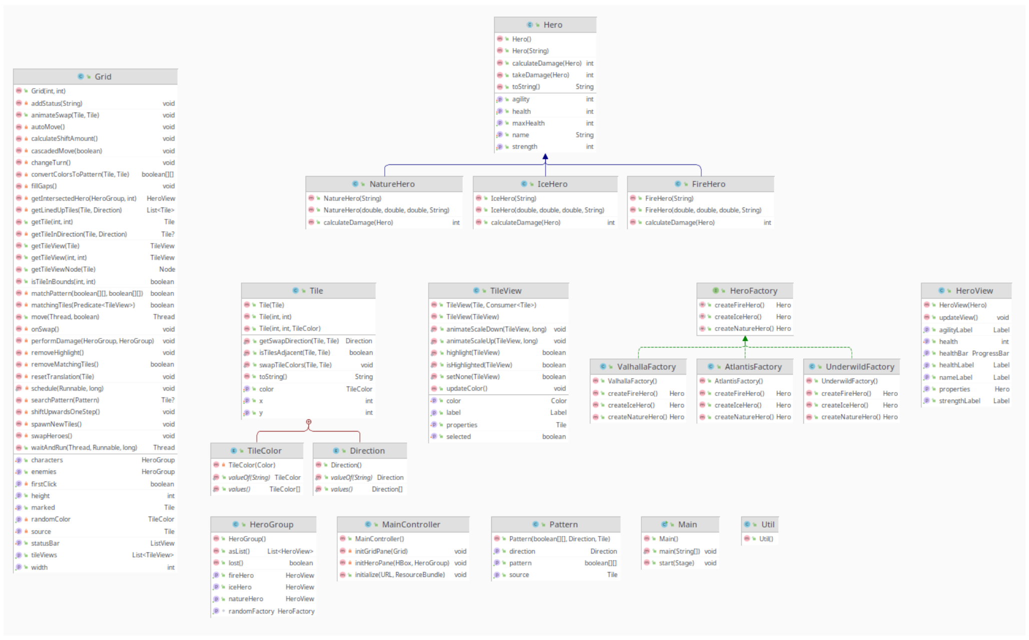
Abstract Factory Pattern

`HeroFactory` is the abstract factory interface, and there are 3 concreate factories which are `ValhallaFactory`, `AtlantisFactory`, and `UnderwildFactory` implements the `HeroFactory`. Factories have 3 methods in order to create heros according to type. Factory methods call the constructor of the `Hero` with given factor values to calculate styled properties.

General damage is calculated in `Hero.java`, and styled (multiplied by 2 or 0.5) in extended hero classes.



General Class Diagram


Grid
Grid(int, int)
addStatus(String) void
animateSwap(Tile, Tile) void
autoMove() void
calculateShiftAmount() void
cascadedMove(boolean) void
changeTurn() void
convertColorsToPattern(Tile, Tile) boolean[][]
fillGaps() void
getIntersectedHero(HeroGroup, int) HeroView
getLinedUpTiles(Tile, Direction) List<Tile>
getTile(int, int) Tile
getTileInDirection(Tile, Direction) Tile?
getTileView(Tile) TileView
getTileView(int, int) TileView
getTileViewNode(Tile) Node
isTileInBounds(int, int) boolean
matchPattern(boolean[][], boolean[][]) boolean
matchingTiles(Predicate<TileView>) boolean
move(Thread, boolean) Thread
onSwap() void
performDamage(HeroGroup, HeroGroup) void
removeHighlight() void
removeMatchingTiles() boolean
resetTranslation(Tile) void
schedule(Runnable, long) void
searchPattern(Pattern) Tile?
shiftUpwardsOneStep() void
spawnNewTiles() void
swapHeroes() void
waitAndRun(Thread, Runnable, long) Thread
characters HeroGroup
enemies HeroGroup
firstClick boolean
height int
marked Tile
randomColor TileColor
source Tile
statusBar ListView
tileViews List<TileView>
width int
Hero
Hero()
Hero(String)
calculateDamage(Hero) int
takeDamage(Hero) int
toString() String
agility int
health int
maxHealth int
name String
strength int
NatureHero
NatureHero(String)
NatureHero(double, double, double, String)
calculateDamage(Hero) int
IceHero
IceHero(String)
IceHero(double, double, double, String)
calculateDamage(Hero) int
FireHero
FireHero(String)
FireHero(double, double, double, String)
calculateDamage(Hero) int
Tile
Tile(Tile)
Tile(int, int)
Tile(int, int, TileColor)
getSwapDirection(Tile, Tile) Direction
isTilesAdjacent(Tile, Tile) boolean
swapTileColors(Tile, Tile) void
toString() String
color TileColor
x int
y int
TileView
TileView(Tile, Consumer<Tile>)
TileView(TileView)
animateScaleDown(TileView, long) void
animateScaleUp(TileView, long) void
highlight(TileView) boolean
isHighlighted(TileView) boolean
setNone(TileView) boolean
updateColor() void
color Color
label Label
properties Tile
selected boolean
HeroFactory
createFireHero() Hero
createIceHero() Hero
createNatureHero() Hero
ValhallaFactory
ValhallaFactory()
createFireHero() Hero
createIceHero() Hero
createNatureHero() Hero
AtlantisFactory
AtlantisFactory()
createFireHero() Hero
createIceHero() Hero
createNatureHero() Hero
UnderwildFactory
UnderwildFactory()
createFireHero() Hero
createIceHero() Hero
createNatureHero() Hero
HeroView
HeroView(Hero)
updateView() void
agilityLabel Label
health int
healthBar ProgressBar
healthLabel Label
nameLabel Label
properties Hero
strengthLabel Label
TileColor
TileColor(Color)
valueOf(String) TileColor
values() TileColor[]
Direction
Direction()
valueOf(String) Direction
values() Direction[]
HeroGroup
HeroGroup()
asList() List<HeroView>
lost() boolean
fireHero HeroView
iceHero HeroView
natureHero HeroView
randomFactory HeroFactory
MainController
MainController()
initGridPane(Grid) void
initHeroPane(HBox, HeroGroup) void
initialize(URL, ResourceBundle) void
Pattern
Pattern(boolean[][], Direction, Tile)
direction Direction
pattern boolean[][]
source Tile
Main
Main()
main(String[]) void
start(Stage) void
Util
Util()