Assignment 4 - Group 8008 - SEM

Joost Verbraeken 4475208
Eric Cornelissen 4447859
Nick Winnubst 4145747
Cornel de Vroomen 4488628
Michael Tran 4499638

Exercise 1 - Code improvements

Removed if-statements

Calc.getRandomIntBetween()

Removed a double If-statement that checked boundary values into one if-statement.
See: `git diff a0f05d2 2672a38`

VerticalPlatform.update() & HorizontalPlatform.update()

Beforehand, there even was an additional method updateEnums() with one if-statement which has moved into the update method and has been modified to remove all pre-existing if-statements in the update() method. This is because the platform is now self-aware of its speed, rendering the whole system present before useless and making the if-statements it used obsolete.
See: `git diff 2672a38 4128f2c`

FileSystem.deleteFile()

There was a if in the deleteFile() method that checked if deleting the file was successful and only logged something in that case. Now the filesystem will always log something, indicating whether or not deleting the file was successful.
See: `git diff b125ab2f 20c0949`

DoodleCamera.update()

The update method in the DoodleCamera previously had an if-statement checking if the camera position must be updated, now we use a little bit more complex maths to determine the new position without an if-statement.
See: `git diff d7c6f97 2ea775e`

AudoManager.Sound.play()

There was an if-statement which checked if there was actually a clip runnen and stopped that clip. This check is not necessary because the clip can always be stopped, whether it is playing or not.
See: `git diff 1e80cfc b14b80a`

SpriteFactory.getDigitSprite(...)

Contained one if-statement and a switch-statement. However, the if-statement performed a check that was included in the switch-statement.
See: `git diff 0d9a04b 800ef86`

Also removed the switch-statement which remained after the refactor described above. This switch-statement was used to return a specific number. Now the method will just return an array of ISprites containing all digits. The only change when using this method is that the index given to the method beforehand must now be appended to the call as [index] (to select the index in the array).
See: `git diff 800ef86 c98118d`

SpriteFactory.getDoodleSprite(...)

Contained a if-statement making a distinction between a right-facing doodle and a left-facing doodle. These have now be implemented in separate methods. The implementation using this method was already written for this change.
See: `git diff d109acc a70fef8`

SpriteFactory.loadISprite(...)

This method had a redundant if-statement that returned something when a certain variable was null, but this was already covered by the try-catch block put in place to catch the exception which would cause the null-value.
See: `git diff 326fccd 9369168`

KillScreen.render()

Contained an if-statement from an old implementation rendering the scene only conditionally. The current implementation of the game doesn't require this as the scene doesn't exist when it shouldn't be active.
See: `git diff dcdbac4 2c8bdd0`

Menu.keyRelease()

Contained an if-statement, but the method was removed entirely because of ambiguous implementation.
See: `git diff be8d5e8 149c899`

PauseScreen.render()

Contained an if-statement from an old implementation rendering the scene only conditionally. The current implementation of the game doesn’t require this as the scene doesn’t exist when it shouldn’t be active.
See: `git diff 215d08c 8ac8673`

World.ScoreBar.ScoreText.getHighestScore()

Contained an if-statement comparing two numbers, now using Math.max().
See: `git diff ab2e0fd 21db308`

World.ScoreBar.PauseButton.mouseClicked()

Contained an if-statement, however the whole button could be implemented using the normal Button implementation.
See: `git diff 21db308 ad92e55`

HighScoreList.initHighScores()

Contained an if-statement checking for null-values, the method which could potentially return the null-value now returns an empty string in these cases.
See: `git diff 88eb3ae 5f94c20`

HighScoreList.loadFromFile()

Contained in if-statement checking the size of the list received from reading the file. This will now be handled by the (already present) try-catch block. As a result, an empty string will be returned so the game can continue.
See: `git diff 24e3b5e d5285d0`

InputManager, Doodle, RegularBehavior, SpaceBehavior, UnderwaterBehavior

Even though there are some if-statements remaining the the latter four, these four classes together have undergone big changes to reduce the number of if-statements in the project (and in particular in the long run.
The implementation of the InputManager regarding key input has been drastically altered. Beforehand it would inform all observers about any key that was pressed. This has changed to a system that knows what keys a observer is interested in, and only informing it about a keypress/release of these particular keys. This implementation has been build using no if-statements by using (Enum)Maps and (Array)Lists.

So why are the classes Doodle, RegularBehavior, SpaceBehavior and UnderwaterBehavior mentioned you might ask? Well, as of now these are the only classes that make use of the Key-input observer method. The change in the system over there has caused a reduction of if-statements in these classes (1 for Doodle and 2 for each Behavior class).
See: `git diff be8d5e8 2e3c8da`

Doodle.setSprite(), RegularBehavior.animate(), SpaceBehavior.animate(), UnderwaterBehavior.animate()

Besides the changes due to the InputManager change, the Behavior classes have gotten a few other if-statements removed. For example the animate() methods (which changed the setSprite() and Doodle as a result). Beforehand this system required 1 if-statement in each Behavior class and 1 in the Doodle class.
The first if-statement (which was in the Doodle class) was removed by using an EnumMap which maps the values that were checked in the if-statement to the correct value.
The second group if-statements (which were in the Behavior classes) where removed by making a clever algorithm that uses Double.compare Math.max to determine the index in an array of length 2 (which comes from the EnumMap).
This system has been made safe by implementing everything centralized in the Doodle, reducing the chance of a faulty implementation. While also ensuring that the attributes used by the system are final and eagerly instantiated.
See: `git diff be8d5e8 2e3c8da`

Intentionally remaining if-statements

Button.mouseClicked()

This method contains an if-statement checking if the click was within certain boundaries, removing this would make the code unnecessarily complex.

ButtonFactory.getButtonFactory()

Contains one if-statement for a singleton implementation which we want to maintain.

FileSystem.getResourceFile()

Contains three if-statements, two of which ensure there are no null-values which we want to maintain as such because the method is public. And changing the other one, which prepends a character conditionally, would make the code unnecessarily complex.

Logger.appendToTextFile()

Contains one if-statement which ensures that the logger only logs when it should log (pending tasks).

LoggerFactory.LoggerFactory()

Contains one if-statement which makes sure the log file is being cleared only when a certain value is true. Removing this would make the code unnecessarily complex.

LoggerFactory.createLogger()

Contains one if-block that makes sure that classes that shouldn't log, don't log by returning a fake logger (an implementation of ILogger without any actual code). Removing this would only introduce more if-statements in each class that uses a Logger.

Calc.getRandomIntBetween()

Contains one if-statement which ensures the input values are correct. Removing this would make using the method unsafe. And replacing with an assert is not recommended because the method is public.

Calc.getRandomDouble()

Contains one if-statement which ensures the input value is correct. Removing this would make using the method unsafe. And replacing with an assert is not recommended because the method is public.

Renderer.drawSprite(...), Renderer.drawSpriteHUD(...)

All contain one if-statement which ensures the input sprite is not null. Removing this would make using the method unsafe. And replacing with an assert is not recommended because the method is public.

Renderer.setGraphicsBuffer(...)

Contains one if-statement which ensures the given graphics buffer is not null. Removing this would make using the method unsafe. And replacing with an assert is not recommended because the method is public.

[ANY CLASS].register() - provided it has a register method

Every register method, used by our service locator, has an if statement checking if the service locator isn't null. Even though the Game would not work at all when the register method is invoked incorrectly, because the game requires all object provided by the service locator to work and it would be clear that something is wrong. We feel it is better to have a if-statement checking for this condition and throwing a (known) error, so it is easier to debug.
It is important to note that the if-statement in each register method is called at most one time, when the game is launched.

HighScore.compareTo(...)

Should return a non-boolean value based on a boolean result, implementation with if-statements is the best solution.

HighScore.equals(...)

A good implementation of equals() must contain an if-statement checking if the given object is null (because it will give a nullpointerException when looking at the attributes). Besides that we use them for early outs (the given object is the current object; Classes are not the same) improving the efficiency.

Intentionally ignored if-statements

Res.setSkin() - To big of a burden for now
Menu.update() - To big of a burden for now

* Of course there are some if- and switch-statements remaining the project which are not discussed (and unavoidably new if/switch-statements where introduced). We simply did not have the time to review all of the code...

Exercise 2 - Teaming up

Shop requirements

Functional requirements

Must haves

- In the menu of the game a scene displaying a shop can be opened.
- In the pause screen of the game a scene displaying a shop can be accessed.
- In the shop, the number of coins a player has earned so far is visible.

Should haves

- Items that can be bought with coins are shown in the shop.
- The prices of items are shown in the shop.
- The upgrades for the trampoline and jetpack can be bought.
- If an item is bought the amount of coins of the player is decreased with the price of the product.
- Items with prices greater than the number of coins the player currently has, can not be bought.

Could haves

- Skins for the doodle can be bought with the coins.
- Different backgrounds can be bought with the coins.

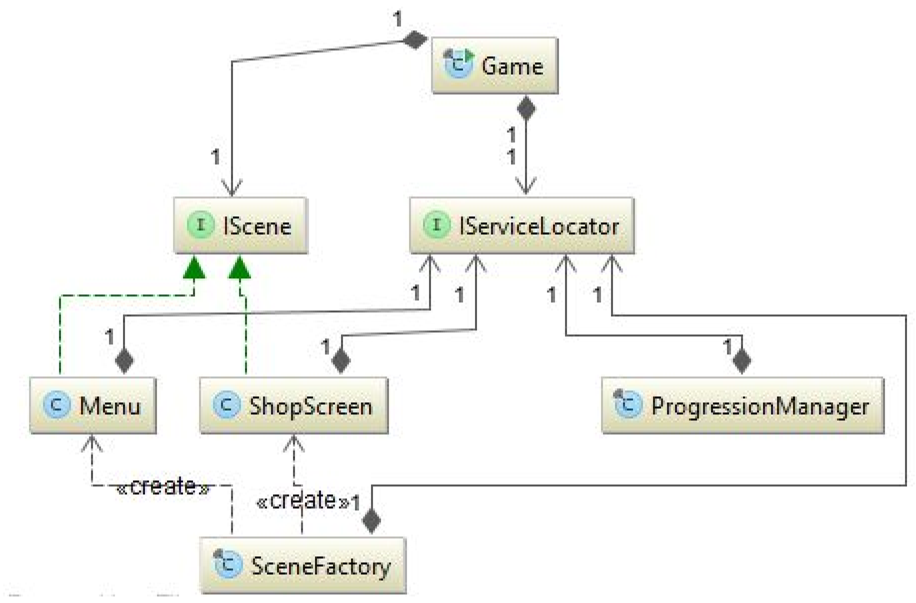
Won't haves

- Buying coins with real money.

Non-functional Requirements

- Besides the provided functionality and services, design constraints need to be included in the requirements specification as well.
- All new features will be implemented in Java 8
- All features will be implemented in the week of 17 October, 2016
- The implementation of the features shall have at least 75% of meaningful line test coverage
- (where meaningful means that the tests actually test the functionalities of the game and for example do not just execute the methods involved)
- No CheckStyle, PMD and FindBugs error will be present in the final version
- Every class, interface, enum, method and field shall be documented using JavaDoc

The UML for the shop:



Exercise 3 - Walking in your TA’s shoes

The full review is too big to put in here, it can be found at `docs/assignment 4/Asteroidsz_Review.pdf`.