Better Snake Game

Dokumentation

Tim Lange tl069

Marcus Maier mm368

Niklas Kieß nk150

Gitrepository:

https://gitlab.mi.hdm-stuttgart.de/mm368/se-2-better-snake

1. Kurzbeschreibung

Das Projekt ist eine Nachahmung des Retro-Spiels Snake.

Das Besondere daran ist, dass zum üblichen Spielprinzip ein paar Eigenschaften hinzugefügt wurden. Die Spielfläche ist wie im Klassiker kariert, mit zusätzlichen Hindernissen in Form von Steinwänden.

Gespielt wird mit den Pfeiltasten und das Ziel ist es, einen möglichst hohen Punktestand zu erreichen, wobei die Schlange möglichst lang werden soll.

Punkte bekommt man durch das Essen von Früchten und Einsammeln von Schatztruhen, welche zufällig auf der Karte auftauchen.

Die Schlange wächst ausschließlich beim Essen von Früchten um jeweils eine Einheit.

2.

Die Main-Methode befindet sich in der passend benannten Klasse „Main“.

Pfad: src/main/java/mainpackage/Main

3.

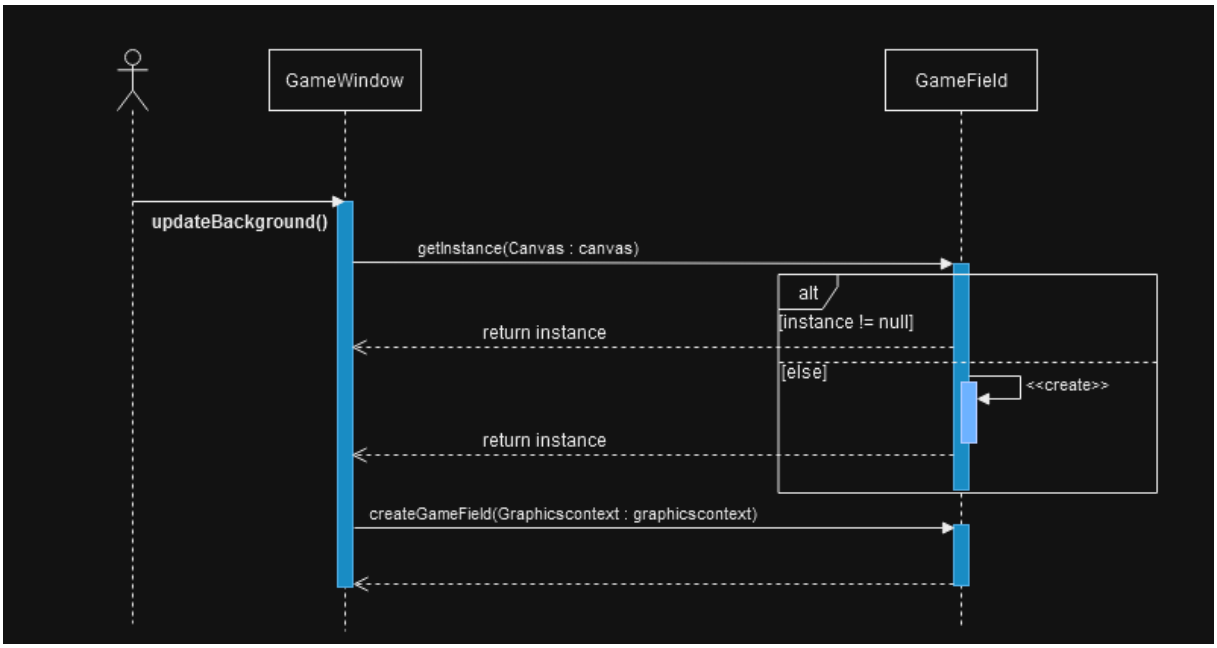
Ein Bug, der die Spielfläche visuell nicht korrekt anzeigt, kann behoben werden, indem man die Main Methode nach Schließen des Fensters erneut ausführt.

Unsere Tests blockieren sich manchmal selbst.

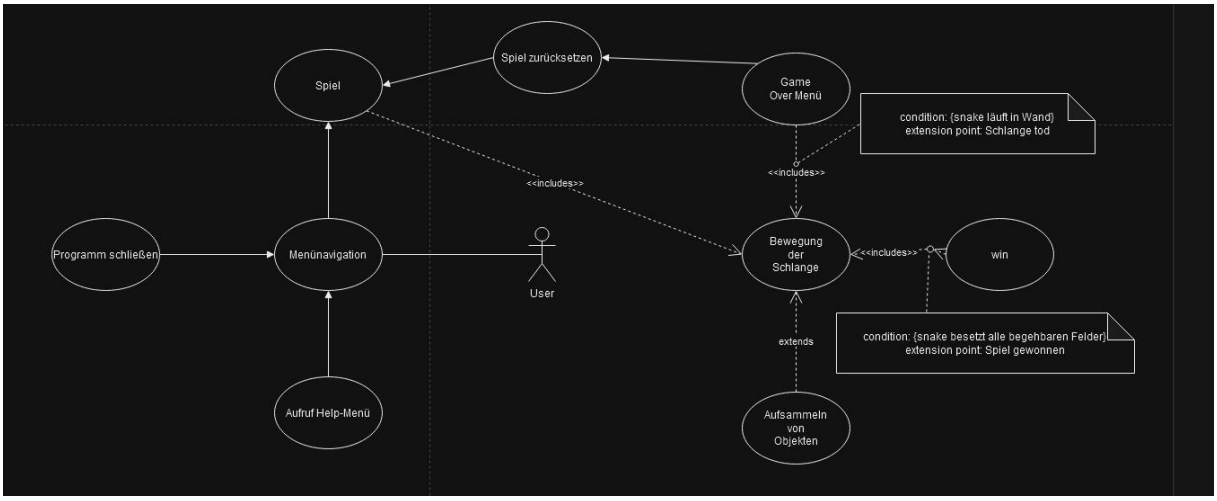
Da wir diesbezüglich keine Lösung haben, muss jeder Test einzeln ausgeführt werden.

4.

Sequenzdiagramm:



Use-Case-Diagramm



UML-Klassendiagramm:

Siehe Ordner src/main/resources

5. Stellungnahme

Interfaces:

IObject: src/main/java/Environment/IObject

AObject: src/main/java/Environment/AObject

IShape: src/main/java/Environment/Obstacle/IShape

AShape: src/main/java/Environment/Obstacle/AShape

In unserem Projekt haben wir es so gehandhabt, dass wir ein Interface und für dieses eine abstract Klasse haben.

Dies haben wir genutzt, da manche Funktionen identisch sind, aber für mehrere Objekte gebraucht werden, weswegen sich eine abstract Klasse gut dafür eignet.

Dokumentation und Testfälle

Wir haben für jede Methode, welche kein Getter oder Setter ist, ein eigenes JavaDoc angelegt.

Zusätzlich haben wir dafür auch Tests in JUnit erstellt.

Diese sind in den Unterordnern des Ordners „test“ vorhanden.

Die Tests bestehen aus einzelnen Testklassen, die jeweils einer zu testenden Klasse zugeordnet wurden.

Logging/Except

Der Beginn jeden Pfades mit src/main/java/ wird zu Gunsten der Übersicht bei den Folgenden Pfaden gedanklich jeweils am Anfang hinzugefügt.

- AShape: Environment/Obstacle/AShape

- ObstacleFactory: Environment/Obstacle/ObstacleFactory
- GameField: Environment/GameField

- GameWindow: Management/Interface/GameWindow
- Score: Management/Interface/Score
- UiManager: Management/Interface/UiManager
- SnakeImageFactory: Management/SnakeManagement/SnakeImages /SnakeImageFactory
- Snake: Management/SnakeManagement/Snake
- GameManager: Management/GameManager
- ObjectManager: Management/ObjectManager

Wir haben für jede Klasse, die relevant für den aktuellen Stand des Spiels ist einen eigenen Logger erstellt.

Diese wiederum senden die wichtigsten Daten in die Konsole.

GUI

Unsere GUI-Elemente sind in dem Ordner src/main/resources vorhanden und sind JavaFx-Elemente.

Sie werden in dem Order src/main/java/ManagementInterface/Scenes verwaltet.

Insgesamt werden vier Szenen verwendet, welche vom Hauptmenü zum Hilfsmenü, dem tatächlichen Spiel, bis hin zum GameOverScreen gehen.

Threads

Klasse: src/main/java/Management/Gamemanager
Zuständige Variable: timerTask (Zeile 46)

Dieser Thread startet unser Ticksystem, welches dafür zuständig ist, das System einheitlich laufen zu lassen. (Bewegung der Schlange und Kollisionserkennung, Erstellung von Wänden etc.)

Streams

Klasse: src/main/java/Management/ObjectManager

Zuständige Methode: randomCoordinate() (Zeile 76)

Unser Stream überprüft ob sich die Position der Schlange, mit der einer Frucht überlappt.
Je nachdem generiert er bei einer bestehenden Überlappung eine neue Koordinate oder bei keiner vorhanden eben nicht.