System Design Document

for dodger game

Authors: Isak Almeros, Tobias Engblom, Viktor Sundberg, Irja Vuorela, Olle Westerlund
30-09-2020
Version 2

1 Introduction

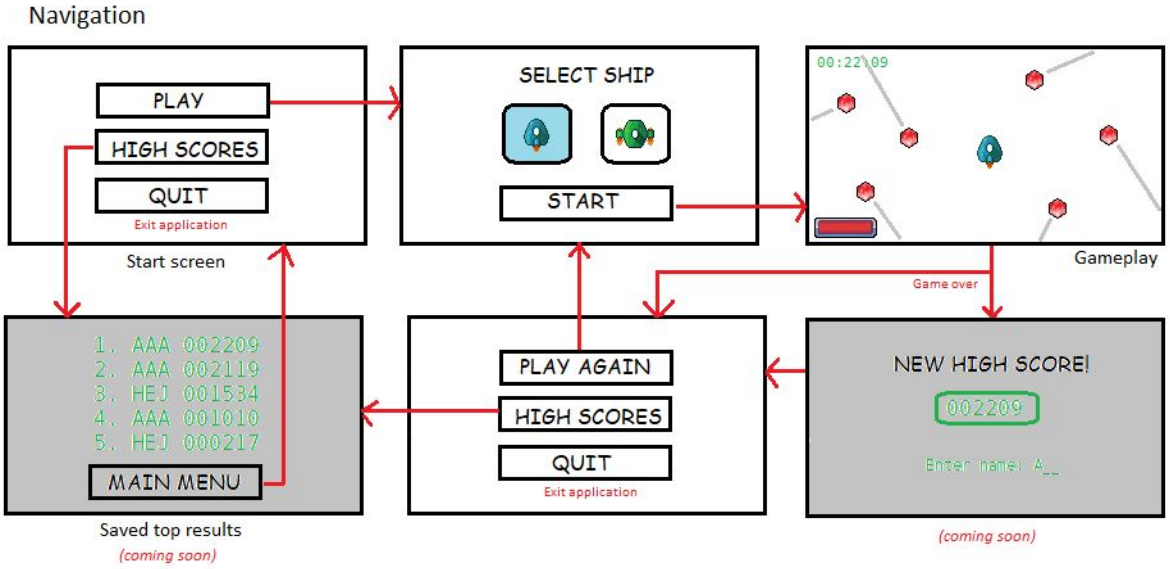
The application is a so-called dodger game where the user controls a small spaceship to avoid different types of objects that's flying about on the playing field. The user gets points based on how long he/she can survive. There are also objects, so-called Power ups, that the player can pick up to gain different abilities or benefits. Examples for power ups are that the player will get a shield around the spaceship so that no damage is taken on the next hit or that you regain some health that you have lost. The application will save a list of high scores with the ten best times.

1.1 Definitions, acronyms, and abbreviations

Unit test: a unit test tests a specific functionality in the code and determines a specific behaviour or state.
HP: Health point - A value representing the player's remaining life.
High score: A list with the top scoring rounds a player has played.
Power up: A game object the player can pick up that gives a temporary advantage.
Character: The entity controlled by the player during the game.
Game Over: The player's hp reaches zero and the game round is therefore ended.
Wraparound: Spelet har inga väggar. När man åker utanför spelkartan flyttas man till motsatt sida och fortsätter i nuvarande färdriktning. (ex Snake) The game has no walls. If the player moves his or her character off screen the character appears on the opposite side and continues in the same direction (e.g. Snake).
Laser beam: A laser beam that fills the playing field from either top to bottom or from side to side, forcing the player to utilize the wrap around to avoid taking damage.
Debuff: A game object that gives the player a disadvantage.

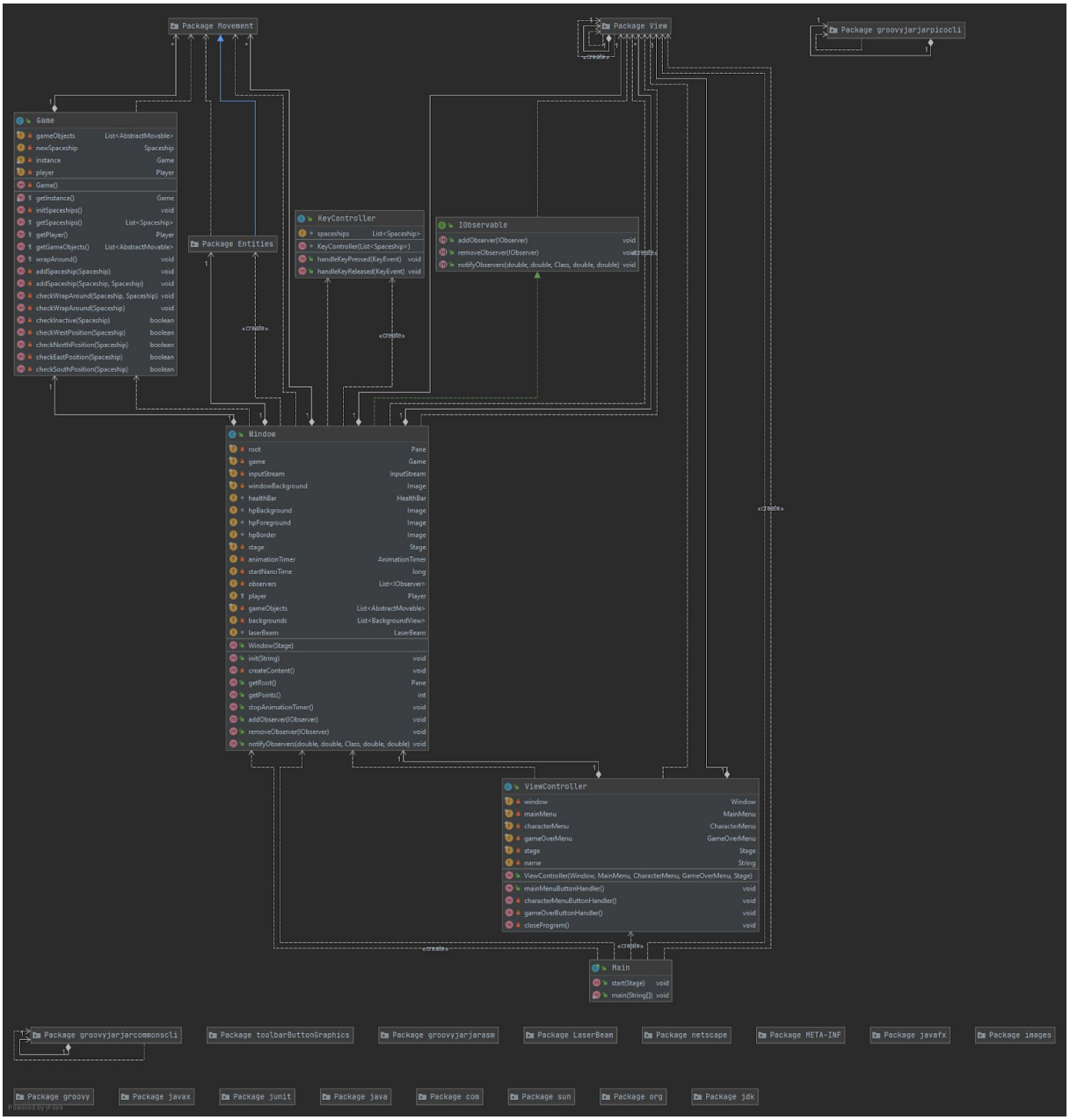
2 System architecture

When the application starts, the user sees a home screen with three possible options. The user can select either to start a new game, look at the current hich scores or quit the application. If the user selects to create a new game he/she will see another view where the user can select different kinds of spaceships and a play button to start the game. The game will then start with the players selected spaceship in the middle of the screen and with asteroids flying in from the sides. When the player's health points reach zero the game is over and the user is met by a game-over view that shows the player's total points for the game. In this view the player can select either to play again right away with the same spaceship, look at the current high scores or return to the main menu. If the player's points is higher than any of the current runs in the high score list the player can type in his/hers name before the game-over view is shown. The list over the top scores are saved localy.

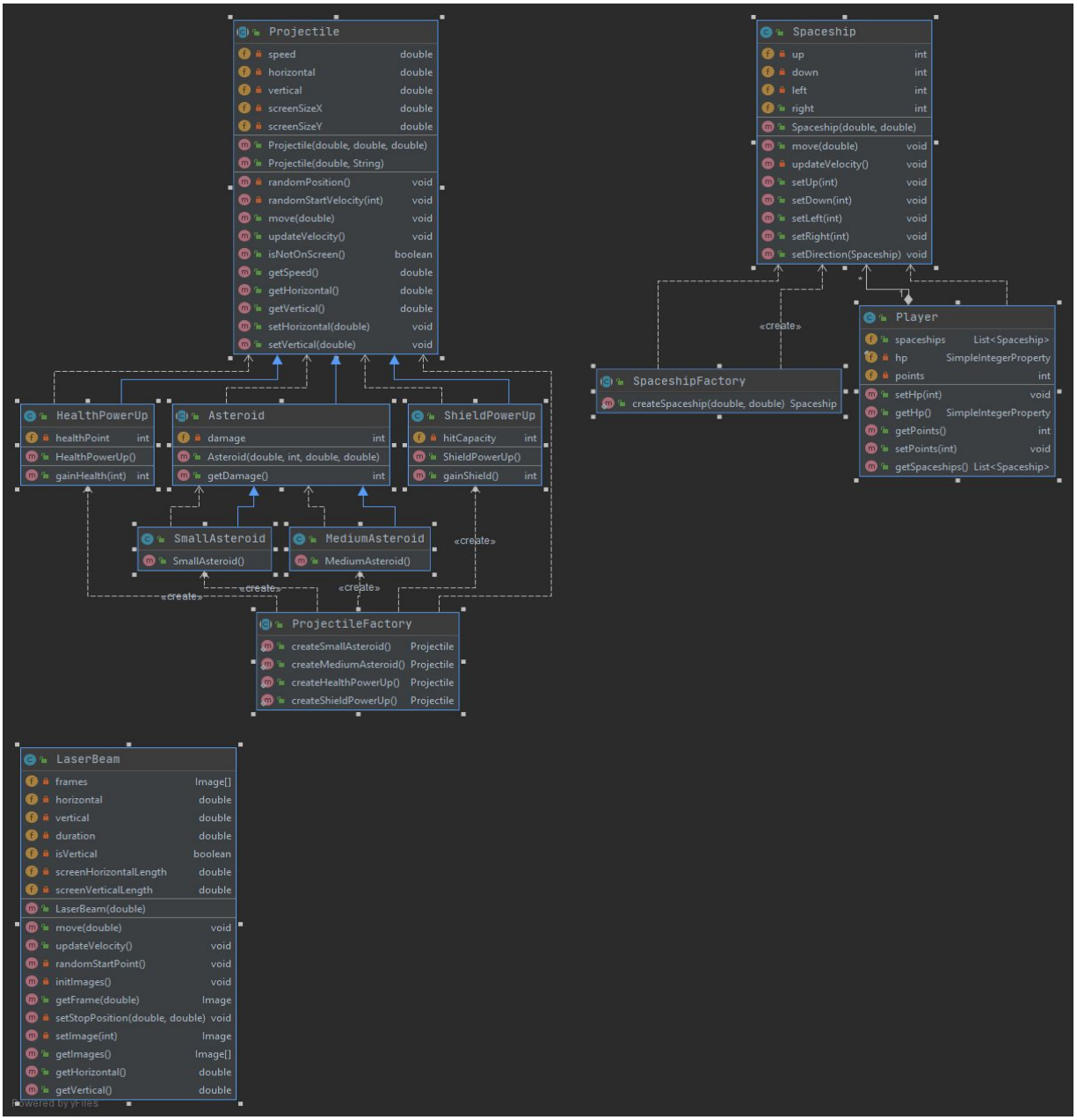


3 System design

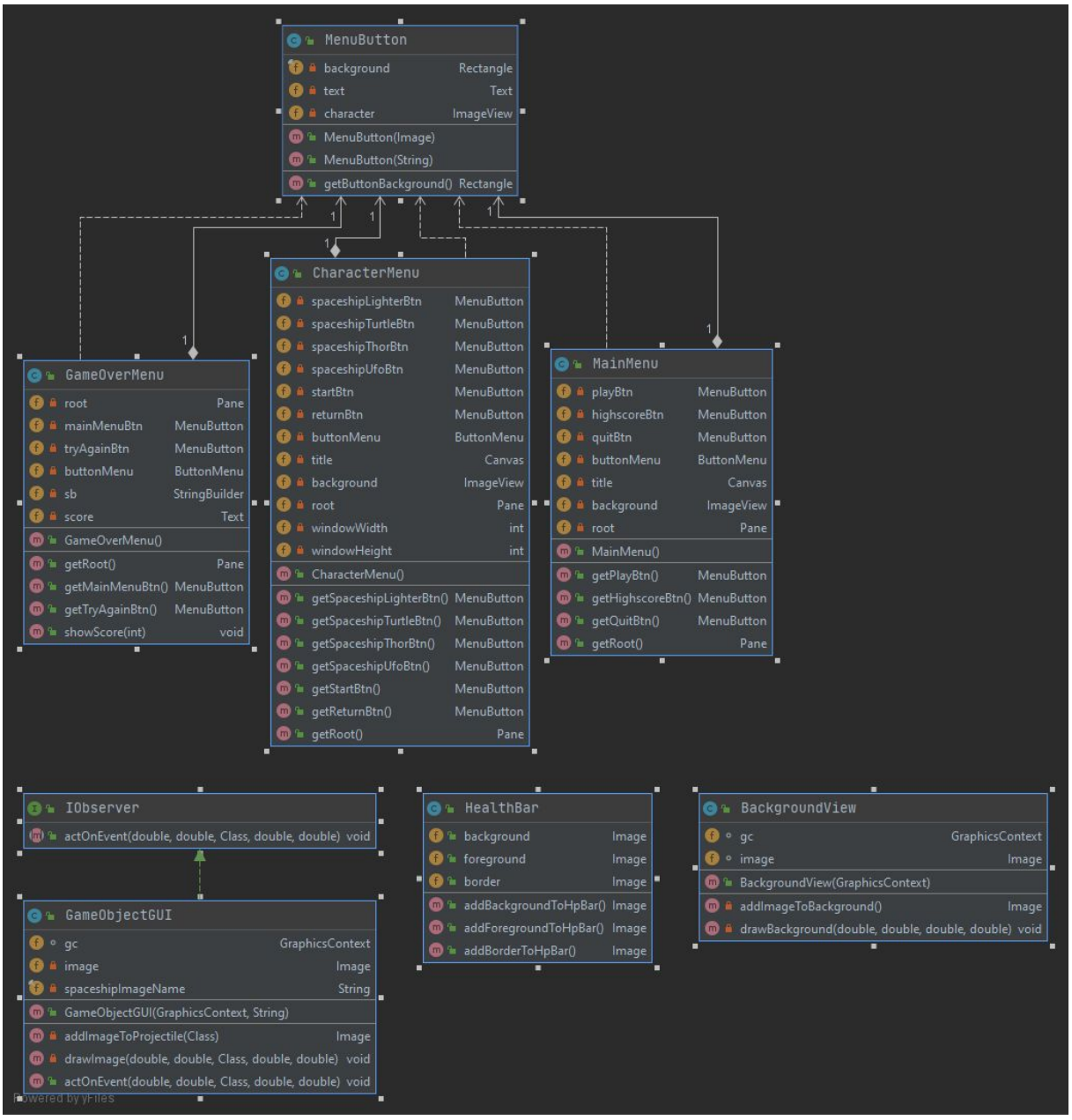
Design model over the top view from the folder java.



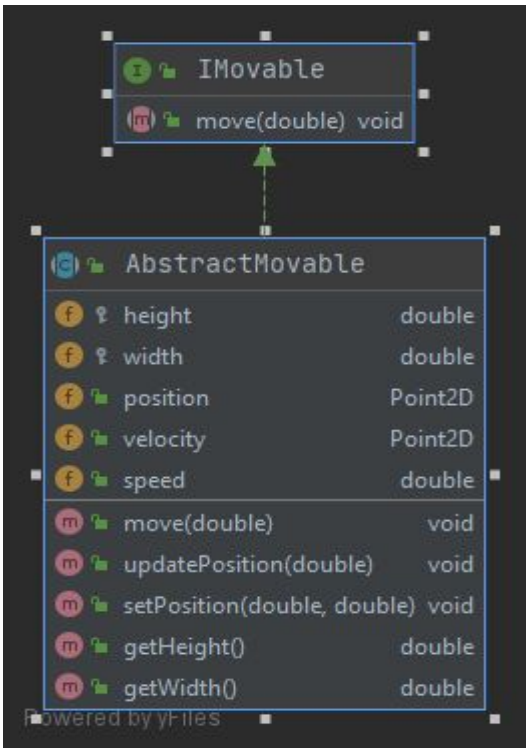
Entities.


Projectile
speed double
horizontal double
vertical double
screenSizeX double
screenSizeY double
Projectile(double, double, double)
Projectile(double, String)
randomPosition() void
randomStartVelocity(int) void
move(double) void
updateVelocity() void
isNotOnScreen() boolean
getSpeed() double
getHorizontal() double
getVertical() double
setHorizontal(double) void
setVertical(double) void
Spaceship
up int
down int
left int
right int
Spaceship(double, double)
move(double) void
updateVelocity() void
setUp(int) void
setDown(int) void
setLeft(int) void
setRight(int) void
setDirection(Spaceship) void
Player
spaceships List<Spaceship>
hp SimpleIntegerProperty
points int
setHp(int) void
getHp() SimpleIntegerProperty
getPoints() int
setPoints(int) void
getSpaceships() List<Spaceship>
SpaceshipFactory
createSpaceship(double, double) Spaceship
«create»
HealthPowerUp
healthPoint int
HealthPowerUp()
gainHealth(int) int
Asteroid
damage int
Asteroid(double, int, double, double)
getDamage() int
ShieldPowerUp
hitCapacity int
ShieldPowerUp()
gainShield() int
SmallAsteroid
SmallAsteroid()
MediumAsteroid
MediumAsteroid()
«create»
«create»
«create»
«create»
ProjectileFactory
createSmallAsteroid() Projectile
createMediumAsteroid() Projectile
createHealthPowerUp() Projectile
createShieldPowerUp() Projectile
LaserBeam
frames Image[]
horizontal double
vertical double
duration double
isVertical boolean
screenHorizontalLength double
screenVerticalLength double
LaserBeam(double)
move(double) void
updateVelocity() void
randomStartPoint() void
initImages() void
getFrame(double) Image
setStopPosition(double, double) void
setImage(int) Image
getImages() Image[]
getHorizontal() double
getVertical() double

View.


MenuButton
background Rectangle
text Text
character ImageView
MenuButton(Image)
MenuButton(String)
getButtonBackground() Rectangle
CharacterMenu
spaceshipLighterBtn MenuButton
spaceshipTurtleBtn MenuButton
spaceshipThorBtn MenuButton
spaceshipUfoBtn MenuButton
startBtn MenuButton
returnBtn MenuButton
buttonMenu ButtonMenu
title Canvas
background ImageView
root Pane
windowWidth int
windowHeight int
CharacterMenu()
getSpaceshipLighterBtn() MenuButton
getSpaceshipTurtleBtn() MenuButton
getSpaceshipThorBtn() MenuButton
getSpaceshipUfoBtn() MenuButton
getStartBtn() MenuButton
getReturnBtn() MenuButton
getRoot() Pane
GameOverMenu
root Pane
mainMenuBtn MenuButton
tryAgainBtn MenuButton
buttonMenu ButtonMenu
sb StringBuilder
score Text
GameOverMenu()
getRoot() Pane
getMainMenuBtn() MenuButton
getTryAgainBtn() MenuButton
showScore(int) void
MainMenu
playBtn MenuButton
highscoreBtn MenuButton
quitBtn MenuButton
buttonMenu ButtonMenu
title Canvas
background ImageView
root Pane
MainMenu()
getPlayBtn() MenuButton
getHighscoreBtn() MenuButton
getQuitBtn() MenuButton
getRoot() Pane
IObserver
actOnEvent(double, double, Class, double, double) void
GameObjectGUI
gc GraphicsContext
image Image
spaceshipImageName String
GameObjectGUI(GraphicsContext, String)
addImageToProjectile(Class) Image
drawImage(double, double, Class, double, double) void
actOnEvent(double, double, Class, double, double) void
HealthBar
background Image
foreground Image
border Image
addBackgroundToHpBar() Image
addForegroundToHpBar() Image
addBorderToHpBar() Image
BackgroundView
gc GraphicsContext
image Image
BackgroundView(GraphicsContext)
addImageToBackground() Image
drawBackground(double, double, double, double) void
1
1
1
1
1
1

Movement.



Singleton pattern: The application uses a singleton pattern when it creates the model, only one model can be created at a time and after that only that instance is returned if some part of the application needs it.

Factory method: When the application creates different types of projectiles it uses a factory method for this.

Observer pattern: When the ship and asteroids in the model changes their position on the playing field the views are notified and then draws out the new position on the playing field.

The application has a listener method that listens on the player's health point's value and when it reaches zero the game is over.

Open/Closed Principle:
The projectile class is an abstract super class for all the different types of projectiles, this makes it easier to add new different kinds of projectiles that have the same base properties and behavior.

4 Persistent data management

All the images that the application uses is in the folder “resources” which is in the same folder as the source root.

The result of the games, the high scores, are saved locally in a text document.

5 Quality

The code is tested with unit tests with help of JUnit.
Testfolder: DodgerGame/src/test/java

At some times the images won’t load from the resource folder, what we have found is that the images need to be added in the target folder. We don’t know why this needs to be done manually sometimes.

5.1 Access control and security

The application uses no form of access control.

6 References

JavaFX - https://openjfx.io/