Delft University of Technology

Software Engineering Methods

CSE2115

Assignment: 2

Moshiur ...
Thijs Nulle
Adrian ...
Ji Darwish
Zohar Cochavi

December 6, 2019

Exercise 1

In the show function of the AliveScreen class, which extends from App.java, should everything for the game be set up. This can be seen where the first call is to create a player, because player extends from entity, this is actually a call to entity. This is the same with Asteroid where first the Asteroid constructor is called and following from that the constructor of Entity. The Anonymous is the function from which the keyDown handler works, all the inputs will be changed based on that function.

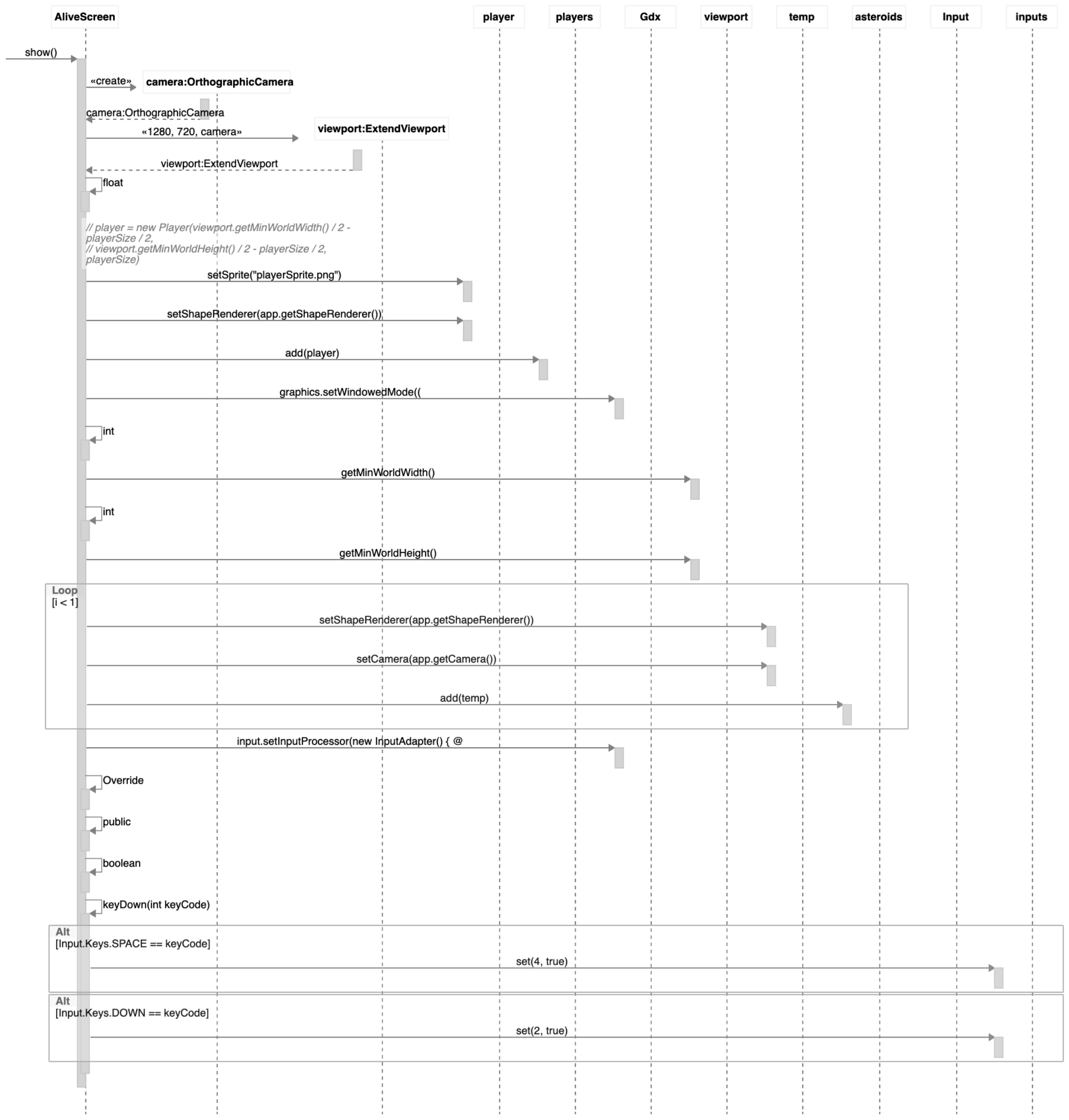
AliveScreen
player
players
Gdx
viewport
temp
asteroids
Input
inputs
show()
«create»
camera:OrthographicCamera
camera:OrthographicCamera
«1280, 720, camera»
viewport:ExtendViewport
viewport:ExtendViewport
float
// player = new Player(viewport.getMinWorldWidth() / 2 - playerSize / 2,
// viewport.getMinWorldHeight() / 2 - playerSize / 2, playerSize)
setSprite("playerSprite.png")
setShapeRenderer(app.getShapeRenderer())
add(player)
graphics.setWindowedMode((
int
getMinWorldWidth()
int
getMinWorldHeight()
Loop
[i < 1]
setShapeRenderer(app.getShapeRenderer())
setCamera(app.getCamera())
add(temp)
input.setInputProcessor(new InputAdapter() { @
Override
public
boolean
keyDown(int keyCode)
Alt
[Input.Keys.SPACE == keyCode]
set(4, true)
Alt
[Input.Keys.DOWN == keyCode]
set(2, true)

Figure 1: AliveScreen.java::show()

In the player update class, the main objective is to update the players values, position and velocity, and update all of its bullets. Within the method, first the player values are updated. After that there is a check if the spacebar is pressed, if it is a new bullet is added to all of its bullets. Then the bullets are being updated one by one.

The asteroid render method first has to create the identity matrix where it saves the current state of the window. Then it sets all the variables for drawing and starts the drawing. Then it draws a polygon with all the vertices and when it is done it ends the drawing phase.

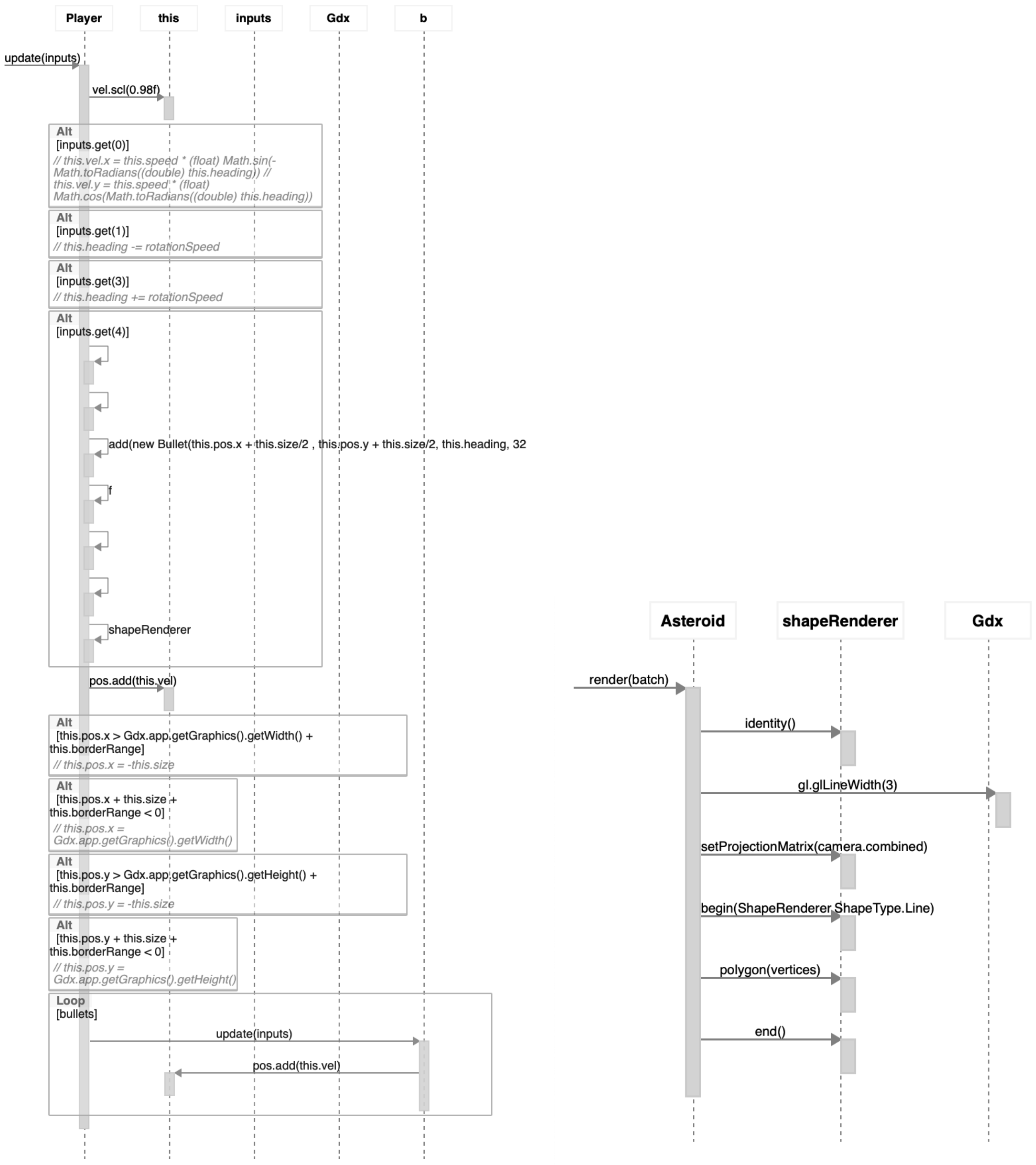
Player
this
inputs
Gdx
b
update(inputs)
vel.scl(0.98f)
Alt
[inputs.get(0)]
// this.vel.x = this.speed * (float) Math.sin(-Math.toRadians((double) this.heading)) // this.vel.y = this.speed * (float) Math.cos(Math.toRadians((double) this.heading))
Alt
[inputs.get(1)]
// this.heading -= rotationSpeed
Alt
[inputs.get(3)]
// this.heading += rotationSpeed
Alt
[inputs.get(4)]
add(new Bullet(this.pos.x + this.size/2 , this.pos.y + this.size/2, this.heading, 32
f
shapeRenderer
pos.add(this.vel)
Alt
[this.pos.x > Gdx.app.getGraphics().getWidth() + this.borderRange]
// this.pos.x = -this.size
Alt
[this.pos.x + this.size + this.borderRange < 0]
// this.pos.x = Gdx.app.getGraphics().getWidth()
Alt
[this.pos.y > Gdx.app.getGraphics().getHeight() + this.borderRange]
// this.pos.y = -this.size
Alt
[this.pos.y + this.size + this.borderRange < 0]
// this.pos.y = Gdx.app.getGraphics().getHeight()
Loop
[bullets]
update(inputs)
pos.add(this.vel)
Asteroid
shapeRenderer
Gdx
render(batch)
identity()
gl.glLineWidth(3)
setProjectionMatrix(camera.combined)
begin(ShapeRenderer.ShapeType.Line)
polygon(vertices)
end()

Figure 2: Player.java::update() and Asteroid.java::render()

Exercise 2

Game Logic

Our game consists of two main classes. Classes that render, and that are renderable.

The moving, and renderable objects in our game are all children of the 'Entity' class. Objects from these set of classes (Bullet, Asteroid, and Player) are renderable and callable by the children of the so-called 'AppScreen' class. This is class extends from libgdx's 'ScreenAdapter' to make it callable by the main object. Using this structure, we are able to change the displayed AppScreen objecs. without changing the main game (in our case the 'App' object) object. (NOTE: There are more children of the AppScreen class. These are essesntially the game-states, but since they include neither game logic, or any authentication they are left out.)

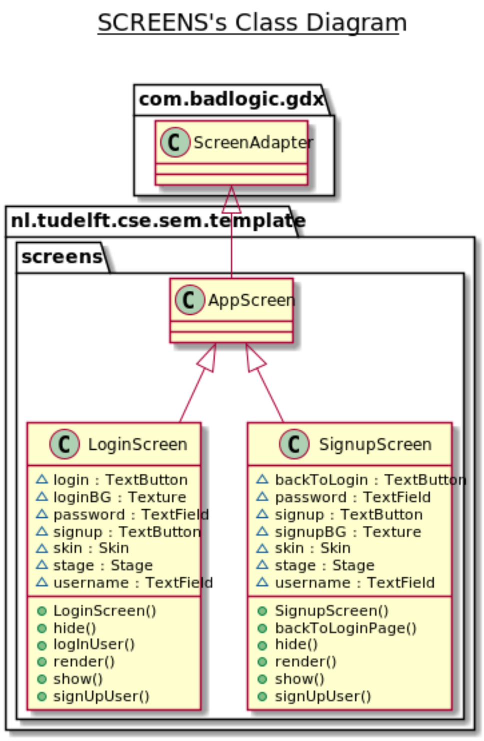

Figure 3: Screen class diagram (NOTE: incomplete)

The classes in the screens package, as displayed above, prompt the user with a login/signin screen. The username and password are retrieved and sent to the database, whose workings are further explained in the next section

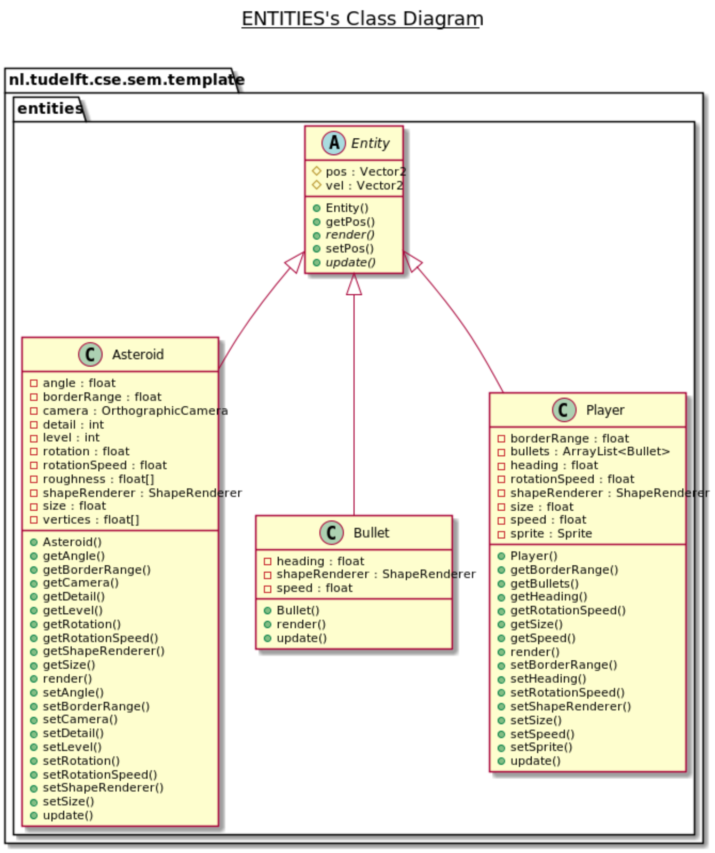

Figure 4: Entities class diagram

The 'AppScreens' simply display the Entities, while the actual game logic is organized by the Entity (sub)classes. Since these objects are moveable and renderable, every Entity has a render function, and getters and setters for the position of the Entity. User input is taken by the ScreenAdapter and passed on to the entities for processing.

Intersting to note is that the shape of all asteroids is generated using libgdx's 'Shape'. This way we can generate the asteroids randomly, and not depend on a couple of sprites.

Database and Authentication

Our database runs on one of the TuDelft servers, and is responsible for the authentication. Authenctication is done using the Spring Authentication Library. This is normally part of the Spring framework, but also available as a separate package. We can connect to the database via the DbConnection object. It send credentials, and retrieves the user data. Then, if successfully authenticated, passes the user data on the the DbConfig object. This object only contains static fields, which contain the user data, so that the program can access them without accessing the database.

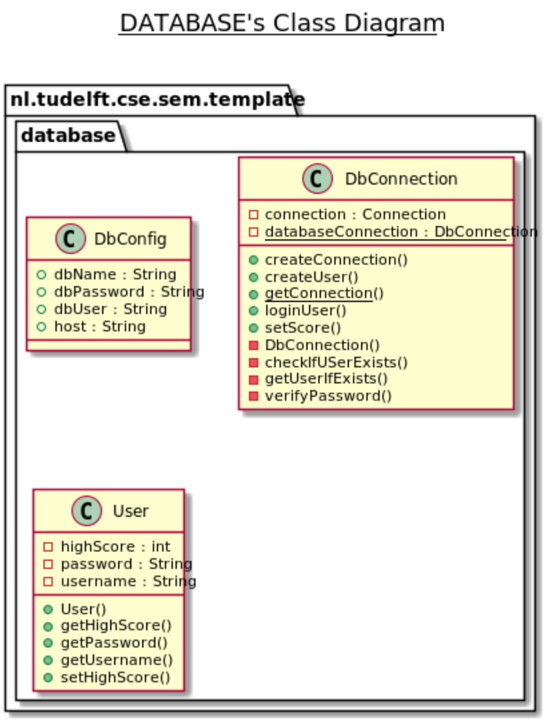

Figure 5: Database class diagram