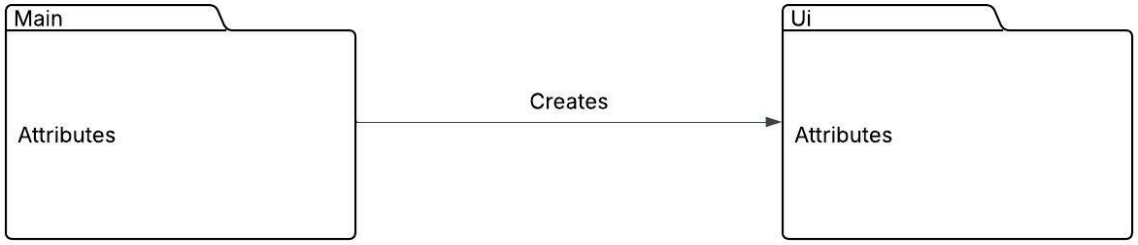
Fish Boid Project
Architectural Design

Author: Jacob Andrew Broomfield
Date: 20th September 2025
Version: 0.2
Status: In progress

CONTENTS

1. STATIC ARCHITECTURE DESCRIPTION

1.1 Package diagram



Main:

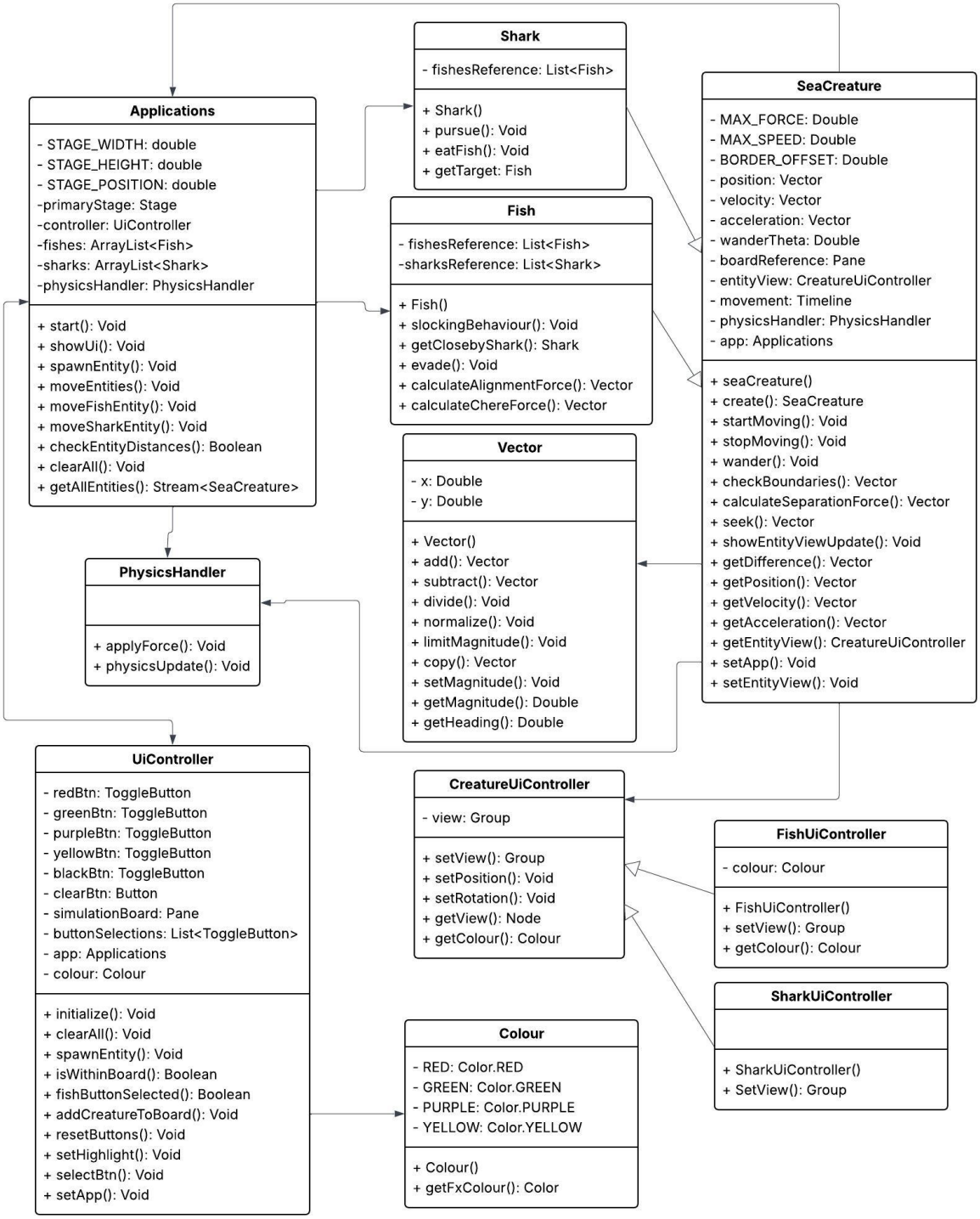
This package contains all the main classes including the Applications class which creates the UI. It also includes the classes for simulating physics like the physics handler and vectors along with the actual main components of the program. This includes the fish and shark class which share the sea creature parent class.

UI:

This package contains all the UI relates classes including the main UI controller, the fish and shark UIs and the colour Enum. There is also a parent UI class for sea creatures too.

2. STATIC DESCRIPTION

2.1 Class diagram



Applications:

Applications is a bridge between the UI and the simulation classes where it creates the UI and orders the simulation behaviours creating and connecting objects with their UI. It controls JavaFX aspects of the program creating the main stage and the UI according to its FXML file. It also sets up objects according to given inputs from the created UI.

UiController:

The UI controller handles all the buttons and panes, connecting the FXML file with functional uses of the buttons and panes created. My UI controller checks for button clicks, highlighting them appropriately. When a mouse click is detected on the main board pane, my UI invokes the application creating an object depending on the currently selected button and spawns a newly created object UI connected to the object by application which UI has a copy of.

Vector:

This is a class for building vector objects which are vital to the object movement in the simulation. It handles various vector maths functions, getting direction and magnitude too. Used to simulate acceleration, velocity and position.

PhysicsHandler:

The physics handler is where the position and velocity vectors are updated to move the object based of the effect of acceleration on the velocity. It also applies various forces given to the acceleration, so outside forces help change the movement of the object.

SeaCreature:

Sea creature is the parent class for fish and shark objects containing all their shared functions and variables. It handles the creature's individual timelines, acceleration/velocity and various movement and entity view methods.

Fish:

Fish class is a subclass of sea creature containing fish specific variables and methods such as position, flee, cohere and alignment for fish boids.

Shark:

Shark class is a subclass of sea creature containing shark specific variables and methods such as pursue, eat and get fish target.

CreatureUiController:

Use to set positions, view and rotations as well as getting object UI colours and views.

FishUiController:

A subclass of creature UI controller setting fish colour and creating the fish group.

SharkUiController:

A subclass of creature UI controller used to create the shark group.

Colour:

This is an Enum of colours containing the JavaFX colour for each selection.

3. DYNAMIC DESCRIPTION

3.1 Pseudocode

US1 & US2 Pseudocode:

```
FUNCTION initialize ():
    SET buttonSelection = ArrayList(redBtn, greenBtn, purpleBtn, yellowBtn, blackBtn)

    FOR button IN buttonSelection:
        button.setOnAction:
            IF button is selected THEN
                unselectAll()
                unhighlightAll()
                selectBtn(button)
                highlightButton(button)
            ELSE
                unselectAll()
                unhighlightAll()
            ENDIF
        END
    ENDFOR
ENDFUNCTION
```

US3 & US7 Pseudocode:

```
FUNCTION spawnFish (mouse click):
    IF x AND y distance is greater than 10 AND fish button selected THEN
        IF x is greater than 0 AND x is less than board size THEN
            IF y is greater than 0 AND y is less than board size THEN
                spawn fish at x,y
            ENDIF
        ENDIF
    ENDIF
ENDFUNCTION

FUNCTION addFishToBoard (Fish):
    fish UI = new UIcontroller
    Fish.setView(fish UI)
    Board.addFishUi
ENDFUNCTION
```

US4 Pseudocode:

```
FUNCTION wander ():
    Vector targetVelocity = velocity.copy
    targetVelocity.setMagnitude(100)
    targetVelocity.add(position)
    double wanderRadius = 50
    double theta = wanderTheta + velocity.getHeading
    double x = wanderRadius * Math.cos(theta)
    double y = wanderRadius * Math.sin(theta)
    targetVelocity.add(Vector(x,y))
    Vector steerForce = targetVelocity.subtract(position)
    steerForce.setMagnitude(MAX_FORCE)
    physicsHandler.applyForce(steerForce)
    wanderTheta += -0.3 + (0.3 + 0.3) * Math.random()
ENDFUNCTION
```

US5 Pseudocode:

```
FUNCTION flockingBehaviour ():
    Vector separationForce = calculateSeparationForce()
    Vector alignmentForce = calculateAlignmentForce()
    Vector cohereForce = calculateCohereForce()
    separationForce.multiply(1.2)
    alignmentForce.multiply(0.8)
    cohereForce.multiply(0.8)
    physicsHandler.applyForce(separationForce)
    physicsHandler.applyForce(alignmentForce)
    physicsHandler.applyForce(cohereForce)
ENDFUNCTION
```

US6 Pseudocode:

```
FUNCTION spawnFish ():
    Vector steerForce = new Vector()
    int count = 0
    FOR SeaCreature otherEntities:
        IF otherEntitt IS NOT this THEN
            IF distance > 0 AND distance < desiredSeparation THEN
                Vector diff = getDifference(position, otherEntity.position)
                diff.normalize()
                diff.divide(distance)
                steerForce.add(diff)
                count++
            ENDIF
        ENDIF
    ENDFOR
    IF count > 0 THEN
        steerForce.divide(count)
    ENDFOR
    IF steerForce.getMagnitude() > 0 THEN
        steerForce.setMagnitude(MAX_SPEED)
        steerForce.subtract(velocity)
        steerForce.limitMagnitude(MAX_FORCE)
    ENDFOR
ENDFUNCTION
```

US8 & US10 Pseudocode:

```
FUNCTION seek ():
    Vector desiredSteerForce = target.copy().subtract(position)
    desiredSteerForce.setMagnitude(MAX_SPEED)
    desiredSteerForce.subtract(velocity)
    desiredSteerForce.limitMagnitude(MAX_FORCE)
    RETURN desiredSteerForce
ENDFUNCTION

FUNCTION pursue ():
    Vector targetPosition = target.position.copy()
    Vector predictedPosition = target.velocity.copy().multiply(10)
    targetPosition.add(predictedPosition)
    physicsHandler.applyForce(seek(targetPosition))
ENDFUNCTION

FUNCTION evade ():
    Vector evadeForce = seek(shark.position).multiply(-1)
    physicsHandler.applyForce(evadeForce)
ENDFUNCTION
```

US9 Pseudocode:

```
FUNCTION eatFish ():
    double distance = getDifference(position, target.position).getMagnitude()
    IF distance LESS THAN 20 THEN
        fishesReference.remove(target)
        boardReference.getChildren().remove(target)
    ENDIF
ENDFUNCTION
```

References

[1] Broomfield, J,A. (2025). “Fish boid nature of code SW”, 1.0, 1st July 2025.

[2] Broomfield, J,A. (2025). “Fish boid timelines SW”, 1.0, 2nd July 2025.

[3] Broomfield, J,A. (2025). “Fish boid Enum SW”, 1.0, 3rd July 2025.

DOCUMENT HISTORY

Version	*Date*	*Changes made to document*	*Changed by*
0.1	17/06/25	Setting up basic document structure.	Jacob Broomfield
0.2	20/09/25	Adding needed titles to the document.	Jacob Broomfield
1.0	23/09/25	Filling and completing the document.	Jacob Broomfield