Assignment 2

TI2206 Software Engineering Methods (2015-2016 Q1)
Date: 25/09/2015
Group Number: 0

Group Members:
- Thomas Baars - tcbaars@gmail.com
- Lars Ysla - yslars@gmail.com
- Boris Schrijver - boris@radialcontext.nl
- Adriaan de Vos - adriaan.devos@gmail.com
- Dylan Straub - d.r.straub@student.tudelft.nl

Table of Contents

Exercise 1 - 20-Time

1. Requirements of Extensions

The requirements of the planned extensions for our Fishy game are as specified below. These requirements are grouped per each planned extension, and outline any additional functional and non-functional requirements for the implementation of the extension. Each requirement is divided into subcategories based on their priority, where requirements marked "Crucial" are crucial for the extension's implementation to be considered a success, and those marked "Non-crucial" are important but will only be implemented if time permits.

Extension 1 - Add high score functionality

Crucial Requirements

- The game shall keep track of the current top ten high scores.
- The current highest score shall be displayed in-game.
- The "Game Over" screen shall display the current highest score.
- The application shall have a "High scores" screen.
- The "High scores" screen shall display the recorded high score information.
- The player shall be able to navigate to the "High scores" screen from the "Title" screen.
- The player shall be able to navigate to the "Title" screen from the "High scores" screen.

Non-crucial Requirements

- When the game is over, the player shall be able to input a 3 character nickname to be associated with the achieved score.
- The game shall record the current high scores to a text file.
- The game shall record the nicknames with the associated scores to a text file.
- When the application is launched, the game shall read the recorded high scores from a text file.
- When the application is launched, the game shall read the recorded nicknames with the associated scores from a text file.

Extension 2 - Add sound effects

Crucial Requirements:

- Certain actions shall have associated sounds.
- The audio file format shall be "wav", "au", or "aif" [1].

[1] Java Sound - https://docs.oracle.com/javase/7/docs/webnotes/tsg/TSG-Desktop/html/sound.html

- The audio file shall have a bit depth of 8-bits or 16-bits per sample [1].
- The audio file shall have a sample rate of 8000, 11025, 22050, or 44100 Hz [1].
- The game shall play the associated sound when the certain actions are performed.
- A sound shall play when the player selects an option.
- A sound shall play when a fish is consumed.
- The player shall have the ability to toggle whether or not sounds are played.
- The game shall have an icon displayed to signify sounds are enabled, when sounds are enabled.
- The game shall have an icon displayed to signify sounds are disabled, when sounds are disabled.

Non-crucial Requirements
- A sound shall play when the game is lost.
- A sound shall play when the game is won.
- A sound shall play when a bubble has popped.

Extension 3 - Add background music

Crucial Requirements:
- The game shall have background music.
- The player shall have the ability to toggle whether or not the background music is played.
- The game shall have an icon displayed to signify music is enabled, when music is enabled.
- The game shall have an icon displayed to signify music is disabled, when music is disabled.
- The music shall loop continuously, when enabled.
- The audio file shall be of a media format that is supported by the JMF 2.1.1 FCS implementation[2].

Non-crucial Requirements
- The music shall pause when the game is paused.
- There shall be multiple background music tracks.
- The game shall continuously play though the playlist of background music tracks, when the music is enabled.
- There shall be digital signal processing applied to the music, for example a “tape-stop” effect.

[2] JMF 2.1.1 FCS implementation - http://www.oracle.com/technetwork/java/javase/formats-138492.html

Extension 4 - Multiple enemy types

Crucial Requirements:

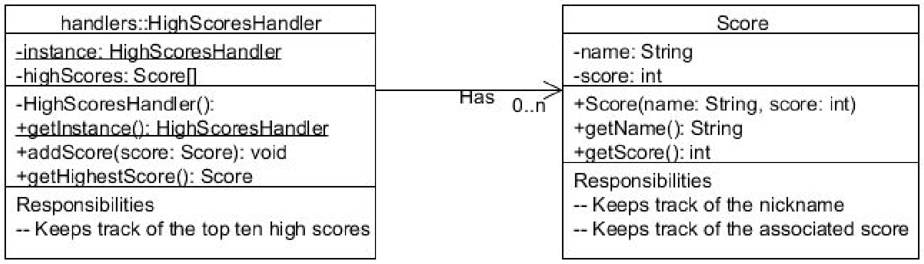
- The game shall have more than one enemy type.
- The base movement speed shall depend on the enemy type.
- The base value shall depend on the enemy type.
- Each enemy type shall have a unique sprite.

2. Analysis and Design of Extensions

The documentation of the analysis and design phase for the planned extensions for our Fish game are included below. The analysis and design documentation produced is divided per each planned extension, and is based on the requirements outlined, using responsibility driven design.

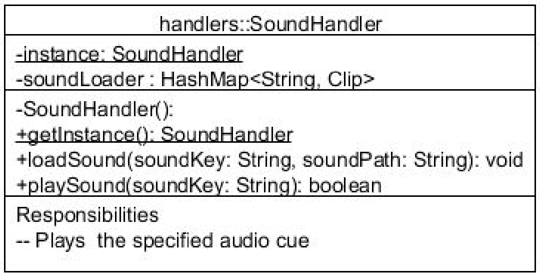
Extension 1 - Add high score functionality

Based on the requirements, the HighScoreHandler and Score classes were designed:


handlers::HighScoresHandler
-instance: HighScoresHandler
-highScores: Score[]
-HighScoresHandler():
+getInstance(): HighScoresHandler
+addScore(score: Score): void
+getHighestScore(): Score
Responsibilities
-- Keeps track of the top ten high scores
Has
0..n
Score
-name: String
-score: int
+Score(name: String, score: int)
+getName(): String
+getScore(): int
Responsibilities
-- Keeps track of the nickname
-- Keeps track of the associated score

Extension 2 - Add sound effects

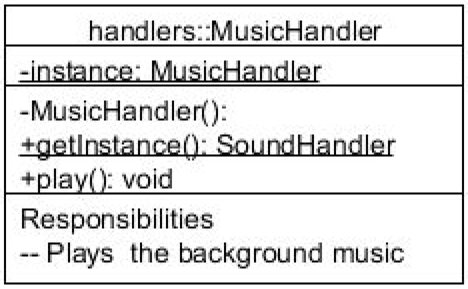
Based on the requirements, the SoundHandler class was designed:


handlers::SoundHandler
-instance: SoundHandler
-soundLoader : HashMap<String, Clip>
-SoundHandler():
+getInstance(): SoundHandler
+loadSound(soundKey: String, soundPath: String): void
+playSound(soundKey: String): boolean
Responsibilities
-- Plays the specified audio cue

which has the main responsibility of playing specified audio cues. This class collaborates with the OptionHandler class to determine whether or not sound is enabled. This class also collaborates with the BackgroundLayer class to display whether or not sound is enabled as well as to listen for when the player want to toggle the sound. This class receives messages from the GUI elements, for example TitleLayer, Player, Trout, and Bubble, to play an audio cue when certain actions are performed.

Extension 3 - Add background music

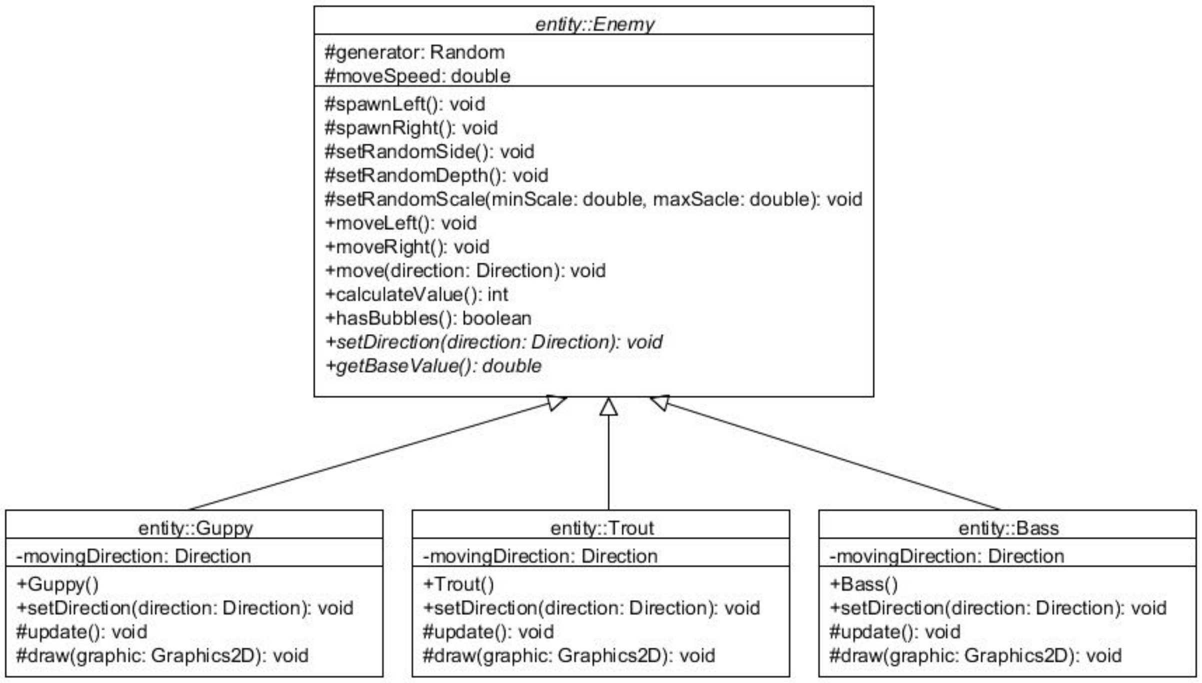
Based on the requirements, the MusicHandler class was designed:



which has the main responsibility of playing the background music. This class collaborates with the OptionHandler class to determine whether or not music is enabled. This class also collaborates with the BackgroundLayer class to display whether or not music is enabled as well as to listen for when the player want to toggle the music.

Extension 4 - Multiple enemy types

Based on the requirements, the multiple enemy types can be implemented through the specialisation of the Enemy class. Where each enemy type extended from the basic enemy, specifies its own special properties, including sprite properties, base movement speed, and base value.


entity::Enemy
#generator: Random
#moveSpeed: double
#spawnLeft(): void
#spawnRight(): void
#setRandomSide(): void
#setRandomDepth(): void
#setRandomScale(minScale: double, maxSacle: double): void
+moveLeft(): void
+moveRight(): void
+move(direction: Direction): void
+calculateValue(): int
+hasBubbles(): boolean
+setDirection(direction: Direction): void
+getBaseValue(): double
entity::Guppy
-movingDirection: Direction
+Guppy()
+setDirection(direction: Direction): void
#update(): void
#draw(graphic: Graphics2D): void
entity::Trout
-movingDirection: Direction
+Trout()
+setDirection(direction: Direction): void
#update(): void
#draw(graphic: Graphics2D): void
entity::Bass
-movingDirection: Direction
+Bass()
+setDirection(direction: Direction): void
#update(): void
#draw(graphic: Graphics2D): void

Exercise 2 - Your wish is my command

1. Requirements of Extensions

The requirements of the additional planned extension for our Fishy game is as specified below. These requirements outline any additional functional and non-functional requirements for the implementation of the extension. Each requirement is divided into subcategories based on their priority, where requirements marked “Crucial” are crucial for the extension’s implementation to be considered a success, and those marked “Non-crucial” are important but will only be implemented if time permits.

Extension 1 - Reformat Entity

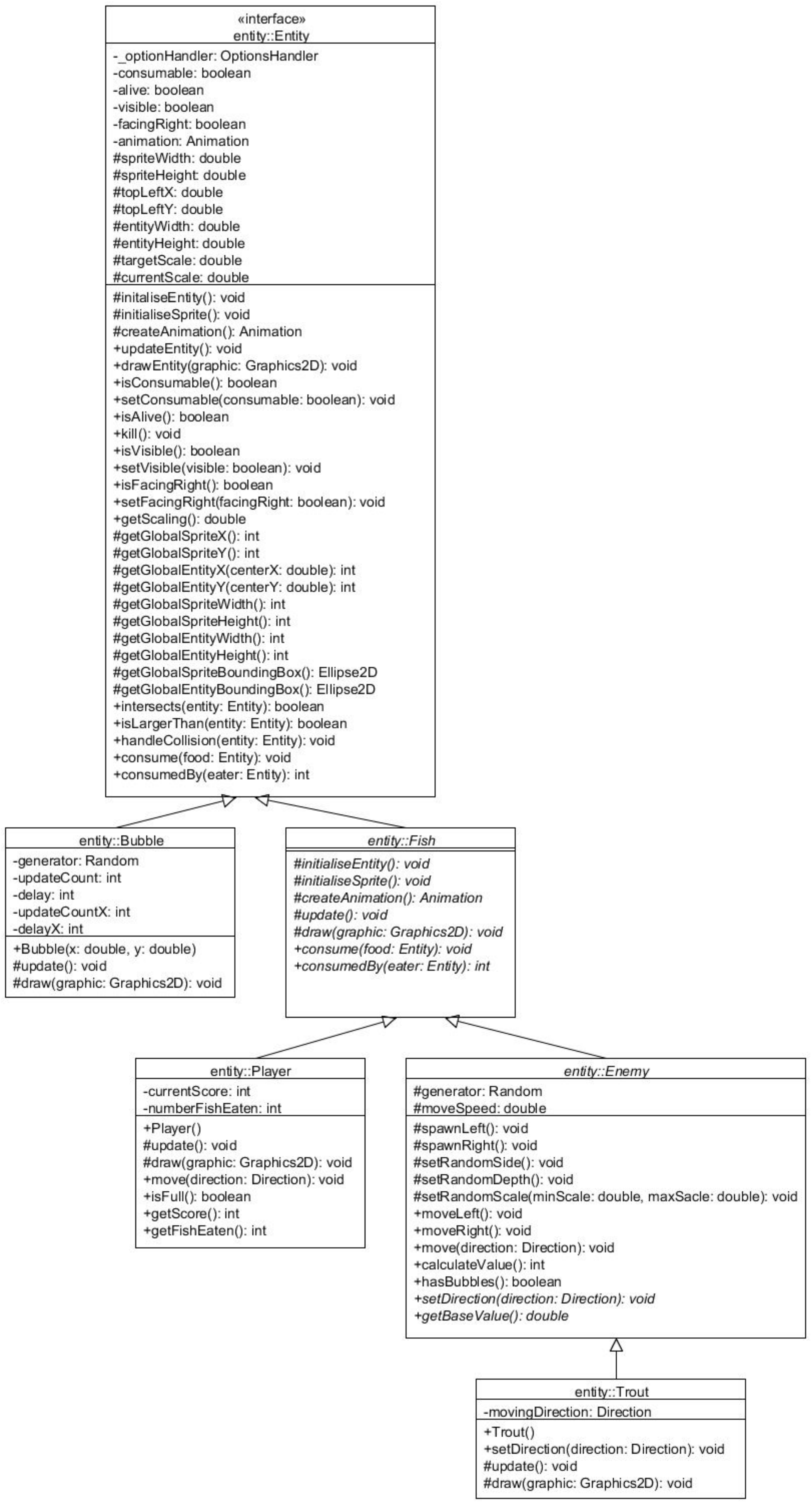
Crucial Requirements:
- The implementation of Entity shall be reformatted to follow the design by contract methodology.

2. Analysis and Design of Extensions

The documentation of the analysis and design phase for the additional planned extension for our Fish game is included below. The analysis and design documentation is based on the requirements outlined, using responsibility driven design.

Extension 1 - Reformat Entity

Based on the requirements the Entity package, was redesigned to follow the design by contract methodology:


«interface»
entity::Entity
-_optionHandler: OptionsHandler
-consumable: boolean
-alive: boolean
-visible: boolean
-facingRight: boolean
-animation: Animation
#spriteWidth: double
#spriteHeight: double
#topLeftX: double
#topLeftY: double
#entityWidth: double
#entityHeight: double
#targetScale: double
#currentScale: double
#initaliseEntity(): void
#initialiseSprite(): void
#createAnimation(): Animation
+updateEntity(): void
+drawEntity(graphic: Graphics2D): void
+isConsumable(): boolean
+setConsumable(consumable: boolean): void
+isAlive(): boolean
+kill(): void
+isVisible(): boolean
+setVisible(visible: boolean): void
+isFacingRight(): boolean
+setFacingRight(facingRight: boolean): void
+getScaling(): double
#getGlobalSpriteX(): int
#getGlobalSpriteY(): int
#getGlobalEntityX(centerX: double): int
#getGlobalEntityY(centerY: double): int
#getGlobalSpriteWidth(): int
#getGlobalSpriteHeight(): int
#getGlobalEntityWidth(): int
#getGlobalEntityHeight(): int
#getGlobalSpriteBoundingBox(): Ellipse2D
#getGlobalEntityBoundingBox(): Ellipse2D
+intersects(entity: Entity): boolean
+isLargerThan(entity: Entity): boolean
+handleCollision(entity: Entity): void
+consume(food: Entity): void
+consumedBy(eater: Entity): int
entity::Bubble
-generator: Random
-updateCount: int
-delay: int
-updateCountX: int
-delayX: int
+Bubble(x: double, y: double)
#update(): void
#draw(graphic: Graphics2D): void
entity::Fish
#initialiseEntity(): void
#initialiseSprite(): void
#createAnimation(): Animation
#update(): void
#draw(graphic: Graphics2D): void
+consume(food: Entity): void
+consumedBy(eater: Entity): int
entity::Player
-currentScore: int
-numberFishEaten: int
+Player()
#update(): void
#draw(graphic: Graphics2D): void
+move(direction: Direction): void
+isFull(): boolean
+getScore(): int
+getFishEaten(): int
entity::Enemy
#generator: Random
#moveSpeed: double
#spawnLeft(): void
#spawnRight(): void
#setRandomSide(): void
#setRandomDepth(): void
#setRandomScale(minScale: double, maxSacle: double): void
+moveLeft(): void
+moveRight(): void
+move(direction: Direction): void
+calculateValue(): int
+hasBubbles(): boolean
+setDirection(direction: Direction): void
+getBaseValue(): double
entity::Trout
-movingDirection: Direction
+Trout()
+setDirection(direction: Direction): void
#update(): void
#draw(graphic: Graphics2D): void