Group 3: Pointy - Final Report

Requirements and Analysis Document for Pointy

Authors: Anton Hildingsson, Erik Magnusson, Joachim Ørfeldt Pedersen, Mattias Oom, Simon Genne

Version: 1.0

Date: 2020-10-23

1. Introduction

Jerk Evert is a topdown 2D game. The player is a simple geometrical shape that navigates a hostile, equally geometrical, world. In this world, the player is attacked by various enemies that shoot different kinds of projectiles at the player. The player itself has no weapon, but instead a set of abilities which (with some creativity) can be used to defeat the enemies. A few of these abilities are "reflection" (reflecting enemy projectiles), "shockwave" (pushing enemies away) and "dash" (making the player invulnerable and very fast for a short period).

The map contains different neutral elements, such as walls and moving walls, which can trap the player, but also be used as cover.

The goal of the player is to defeat all enemies. The game contains multiple levels, and by defeating one level, the player can progress to the next.

1.1. Definitions, acronyms, and abbreviations

- 2D - Two dimensional
- Topdown game - A game that is viewed from above.
- Projectile - Object fired by some entity towards some other entity. Can cause harm to the player or enemies.
- Obstacle - Neutral object placed on the level. Some obstacles may hinder the movement of the player, enemies, and projectiles in the game, while others may cause harm towards the player or the enemies.
- Ability - Can be used by the player or enemies to impact the flow of the game in some way. This can, for example, be by adding a projectile to the game, or by directly impacting the surrounding entities in some way.
- Player - Entity controlled by the user of the Game.
- Enemies - Opponents of the player. Will use their abilities to try to defeat the player.

2. Requirements

2.1. User Stories

Story Identifier: STK001 (FINISHED)

Story Name: Basic Player

Time estimate: 3 days

Description: As a user, I want a basic player character which I can maneuver using WASD or the arrow keys.

Confirmation:

- **Functional:**
    - Can I move the player using keyboard keys?
- **Non-functional:**

  Responsiveness:
    - Does the player respond in a predictable way?

  Availability:
    - Does the player always respond?

  Visibility:
    - Is there a visual indication of the players movement?

Story Identifier: STK002 (FINISHED)

Story Name: Basic enemy

Time estimate: 2 days

Description: As a user, I want a enemy to compete against when playing a game, in order to have a fun gaming experience.

Confirmation:

- **Functional:**
    - Does the enemy respond to user actions?
    - Is the enemy a real danger?
- **Non-functional:**

  Responsiveness:
    - Do the enemies respond in a reasonable way?

  Visibility:
    - Is the enemy actions visible to the user?

Story Identifier: STK003 (FINISHED)

Story Name: Basic map

Time estimate: 2 days

Description: As a user, I want a game to have a basic map in which I can navigate and explore.

Confirmation:

- **Functional:**
    - There is a map with four walls at the edges, which cannot be crossed.
- **Non-functional:**

  Responsiveness:
    - The walls of the map functions in a predictable way (cannot be crossed).

  Visibility:
    - The size and layout of the map is clearly visible to the user.

Story Identifier: STK004 (FINISHED)

Story Name: Ability reflect

Time estimate: 1 day

Description: As a player I want the ability to reflect projectiles, to protect myself and hurt hostile elements. If the ability has a cooldown, it would force me to use it strategically and introduce an interesting gameplay element.

Confirmation:

- **Functional:**
    - Can I click a key to activate this ability?
    - Is the ability unusable during the time of the cooldown?
- **Non-functional:**

  Gameplay:
    - Is there a clear benefit to using the ability?
    - Does the ability enable me to hurt hostile elements by reflecting their projectiles?

  Availability:
    - Is there a clear indiciation to when the ability is available?

Story Identifier: STK005 (FINISHED)

Story Name: Different levels

Time estimate: 5 days

Dessription: As a user, I want a variety of levels, so that the gameplay doesn't become to similair.

Confirmation:

- **Functional:**
    - Can I play on maps that have different structures/enemies/obstacles?
- **Non-functional:**
    - Is there an increasing difficulty level as the game progresses?

Story Identifier: STK006 (FINISHED)

Story Name: Level transition

Time estimate: 4 days

Description: As a player, I'd like to have a way to transition from one section of the game to another, to get a sense of progression.

Confirmation:

- **Functional:**
    - Can I transition to another part of the game, when one part is done?
- **Non-functional:**

  Interactivity:

    - ~~Is the transition interactive, for example, can I move to another part of the map to enter the next game section?~~

Story Identifier: STK007 (FINISHED)

Story Name: Start menu

Time estimate: 4 days

Description: As a user, I want a start menu so I can decide when to start the game.

Confirmation

- **Functional:**
    - Can I click a button to start the game?
    - Can I click a button to quit the game?
    - Is it possible to pause the game?

- **Non-functional:**

  Availability:

  - Is the menu always accessible when the game is started?

  Usability:

  - Is the menu intuitive and easy to use?

Story Identifier: STK008 (FINISHED)

Story Name: Ability dash

Time estimate: 1 day

Description: As a player, I'd like an ability to dash, to avoid dangerous elements. If the ability has a cooldown, it would make the gameplay more interesting, and force me to be more strategic and conservative with the use of this ability.

Confirmation:

- **Functional:**

  - Can I click a key to activate this ability?
  - Is the ability unusable during the time of the cooldown?

- **Non-functional:**

  Gameplay:

  - Is there a clear benefit to using the ability?

  Availability:

  - Is there a clear indiciation to when the ability is available?

Story Identifier: STK009 (FINISHED)

Story Name: Ability shockwave

Time estimate: 4 days

Description: As a player I need an ability to perform a shockwave so that I can push enemies away from me.

Confirmation:

- **Functional:**

  - Can I push a button and have the ability function reliably?
  - Are the enemies pushed away from me when I use the ability?

- **Non-functional:**

Availability:

- Can I see when my ability is available and when it's not?
- Can I easily figure out how my to access my ability?

Usability:

- Is my ability which easily accessible?
- Can I easily figure out how my ability works?
- Can I easily understand the range and limits of my ability?

Story Identifier: STK010 (FINISHED)

Story Name: Bullet enemy

Time estimate: 5 days

Description: As a player, I want to fight against enemies which can shoot bullets for me to avoid, to increase the difficulty of the game and make it more enjoyable to play.

Confirmation:

- **Functional:**
  - Can the enemy shoot bullets at me, the player?
  - Is there a consequence to being hit by the bullets?
- **Non-functional:**

  Responsiveness:

  - Does the enemies aim at me?

  Difficulty:

  - Is the rate of fire and speed of the bullets reasonable?

  Visibility:

  - Are the bullets visible to me?

Story Identifier: STK011 (FINISHED)

Story Name: Missile enemy

Time estimate: 5 days

Description: As a player, I want to fight against enemies which can shoot homing missiles which target me as the player. This would increase the difficulty and propose a different game mechanic, making the game more fun to play.

Confirmation:

- **Functional:**
    - Can the enemy shoot missiles targeted at me, the player?
    - Is there a consequence to being hit by the missiles?
- **Non-functional:**

  Responsiveness:

    - Does the enemies aim at me?
    - Does the missiles react to my movements?

  Difficulty:

    - Are the missiles reasonably possilbe to avoid?

  Visibility:

    - Are the missiles visible to me?

Story Identifier: STK012 (FINISHED)

Story Name: Obstacles

Time estimate: 2 days

Description: As a player, I'd like a set of obstacles to navigate around, to make the gameplay more varied and strategic.

Confirmation:

- **Functional:**
    - Can I collide with obstacles?
    - Can other enteties collide with obstacles?
- **Non-functional:**

  Gameplay:

    - Can I use the obstacles to avoid dangerous elements?

  Visibility:

    - Can I see and navigate around the obstacles?

Story Identifier: STK013 (FINISHED)

Story Name: Spikes

Time estimate: 2 days

Description: As a player, I want there to exist "spikes", or dangerous, static game elements, for me to avoid. This would create more interesting gameplay, especially since also enemies could be hurt by spikes.

Confirmation:

- **Functional:**
    - Can I, and enemies, collide with spikes?
    - Does the spikes hurt me, or the enemies, when hit?
- **Non-functional:**

  Gameplay:

    - Can I use the spikes to my advantage?

  Visibility:

    - Can I see the spikes, to know how to navigate around them?

Story Identifier: STK014 (IN PROGRESS)

Story Name: Scoring system

Time estimate: 6 days

Description: As a user, I want a way to keep track of my score, so that I get motivated to improve at the game.

Confirmation:

- **Functional:**
    - ~~Can I get points by performing certain actions throughout the game?~~
    - Can I see my score while playing?
    - Can I keep track of how many points I've had in previous playthroughs?
- **Non-functional:**
    - ~~Does the difficulty of an action affect how many points that action is worth?~~

Story Identifier: STK015 (FINISHED)

Story Name: Music

Time estimate: 14 days

Description: As a user, I want background music, so that I get a more immersive gameplay experience.

Confirmation:

- **Functional:**

- Can I hear music as I am playing the game?

Story Identifier: STK016 (FINISHED)

Story Name: Sound effects

Time estimate: 14 days

Desscription: As a user I want sound effects so that I get a more immersive gameplay experience.

Confirmation:

- **Functional:**
  - Are there sound effects that reflect the players actions?
  - ~~Are there sound effects that reflect the actions of the enemies?~~
- **Non-functional:**
  - Do sound effects indicate which abilities have been used?

Story Identifier: STK017

Story Name: Slowmotion

Time estimate: 4 days

Description: As a player, I need an ability that lets me slow the environment around me so that I can escape dangers more easily.

Confirmation

List all acceptance criteria; you should be able to test/confirm these.

- **Functional**

- Can I push a button and have the ability function reliably?
- Is everything around me slowed down when using the ability?

- **Non-functional**

  Gameplay
  - Is there a clear benefit to using the ability?

  Availability
  - Can I see when my ability is available and when it's not?
  - Can I easily figure out how my to access my ability?

  Usability
  - Is my ability easily accessible?

- Can I easily figure out how my ability works?

Story Identifier: STK018 (FINISHED)

Story Name: Moving obstacles

Time estimate: 1 days

Description: As a player, I want there to exist moving game elements which would make for a more dynamic gameplay.

Confirmation

- **Functional**

- Do obstacles move?
- Do obtacles stop entities only when they are in that exact position?

- **Non-functional**

  Gameplay

  - Can I use the moving obstacles to my advantage?

  Visibility

  - Can I understand how the obstacle moves?

2.2. Definition of done

For a user story to be considered to be done, the following criteria have to be fullfilled:

- All acceptence criteria of that user story are satisfied.
- All tests have been passed.
- All visual features of the user story have been added to the GUI.

2.3. User Interface

GUI sketch

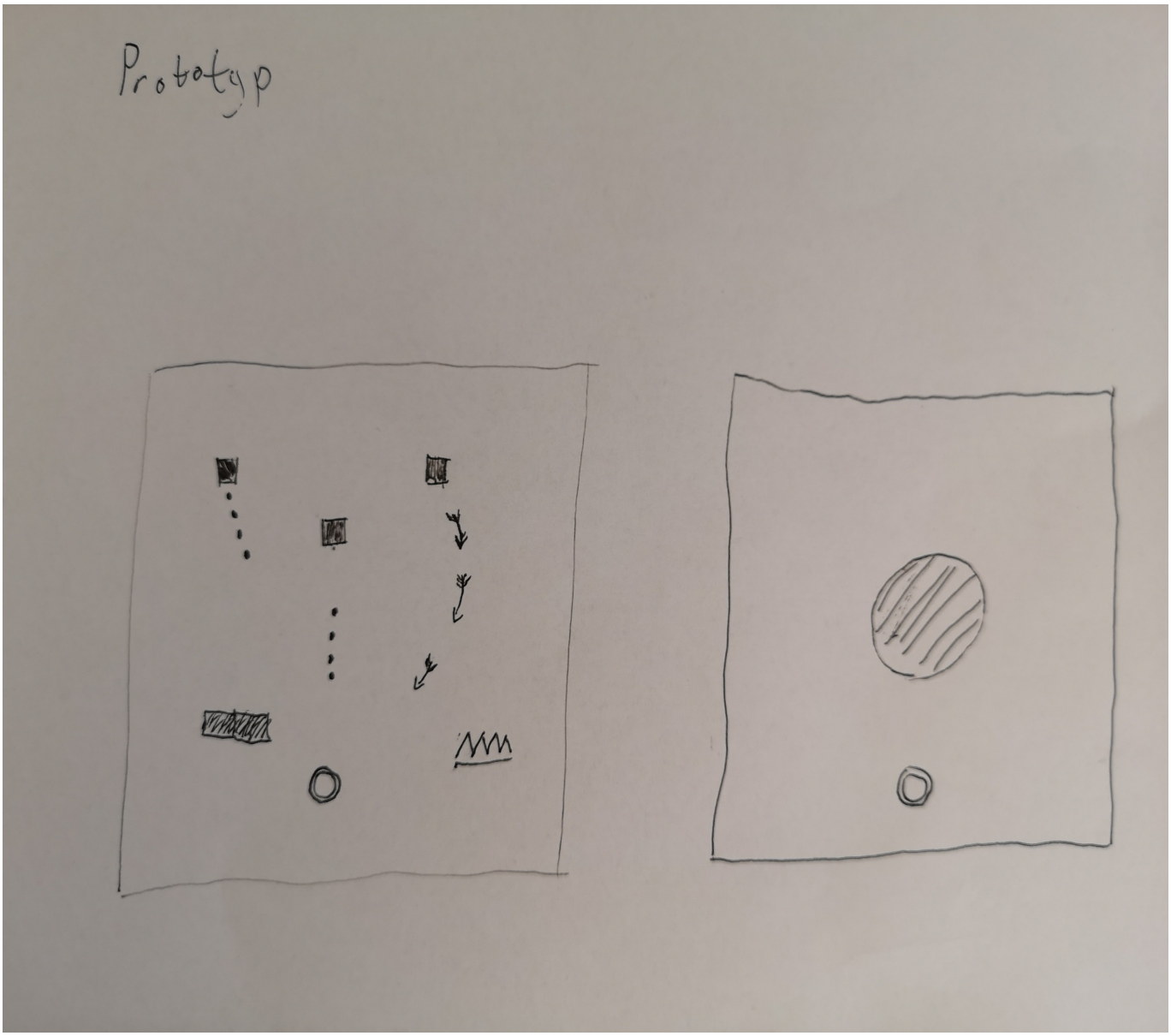


Image 1

A player (circle) with a small ring around. The ring indicates the state of one of the player's abilities.

Three squares represent enemies opposing the players. Each enemy is shooting some kind of projectile.

The small dots represent simple bullets, while the arrows represent some kind of homing missiles.

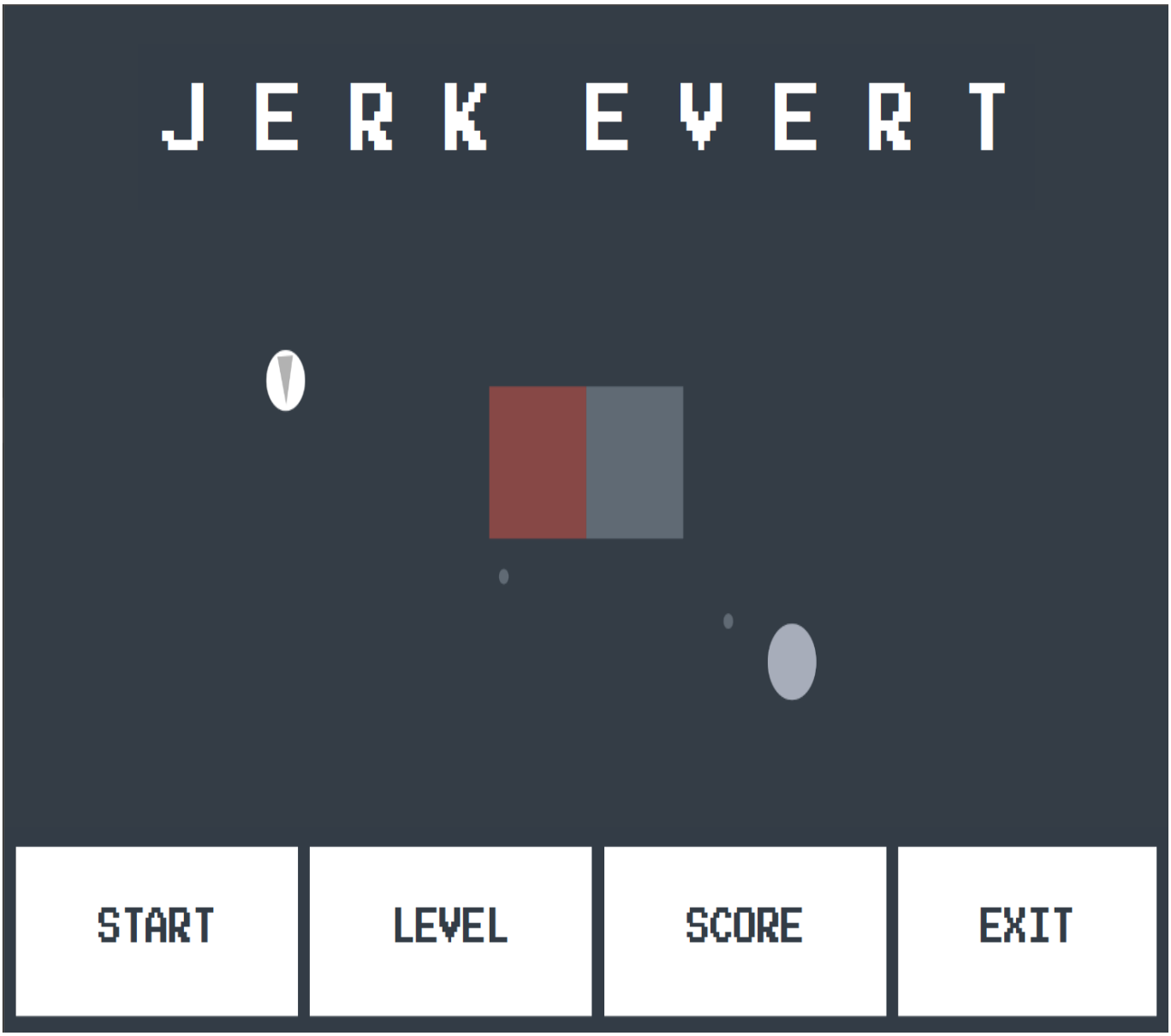
The rectangle represent a wall, an obstacle for both players and enemies.

The spikes represent a dangerous element which can hurt both players and enemies.

Image 2

This image represent when the player moves from one level to the next. When the player has completed their objective, a gate is opened at the center of the map. This gate can be used to enter the next level.

Start screen

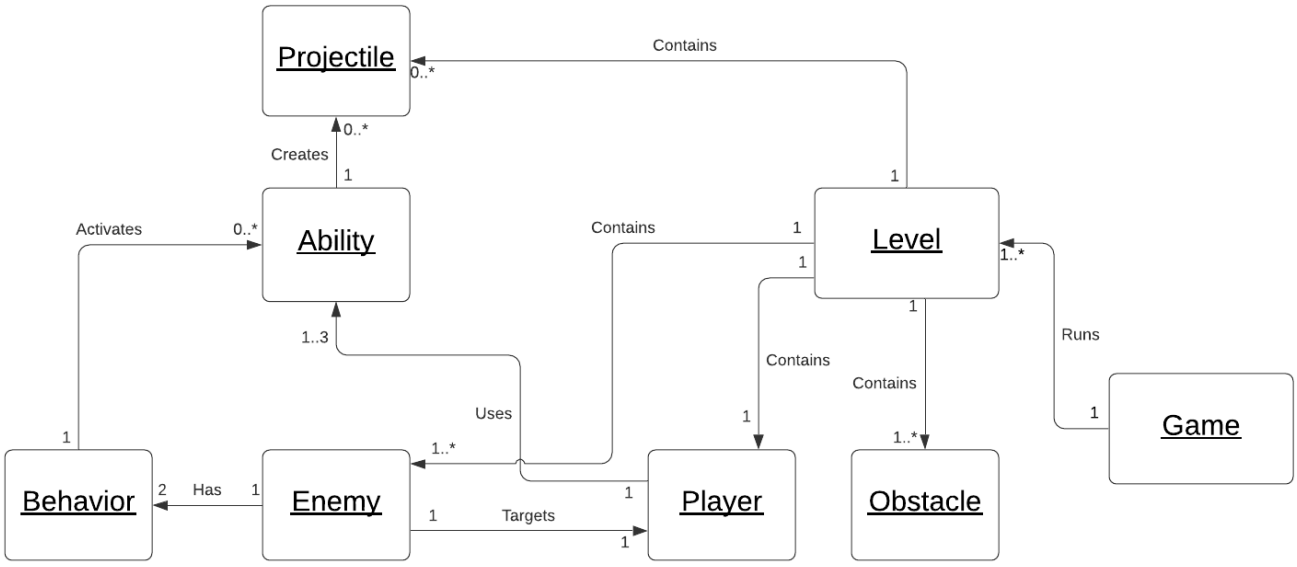


Displayed when the game is launched.

3. Diagrams

3.1 Domain Model

UML of domain model



3.1.1 Class responsibilities

Explanation of responsibilities of classes in diagram. (More details in the actual class) Stepping into a package adds an indentation.

controller (package)

event (package)

AbilityActionEvent

Event which is sent when an ability action is activated or finished.

AbilityAcitonHandler

The event handler for ability action events.

AbilityActionEventListener

Event listener for ability action events.

IAbilityActionEvent

Simple abstraction for ability action events.

gameLoop (package)

gameLoop

Calls an update method each loop iteration at a desired fps.

IGameLoop

Simple abstraction for game loops.

Action

Simple interface for defining lambda expression with no arguments and no arguments.

GameWindowController

This class handles the model and view components of the game, starts the game loop, and ties all components together. Without the game window controller, there >> would be no game.

KeyboardInputController

Handles keyboard input from the user and ties different keyboard presses to a specific method call.

MouseInputController

Handles mouse input from the user and ties different mouse actions to specific method calls.

model (package)

ability (package)

action (package)

AbilityAction

Abstract implementation of ability action interface.

IAbilityAction

An abstraction for defining which actions should be performed on a level when an ability is applied.

Ability

Abstract implementation of the ability interface. This simplifies creating new abilities by implementing the cooldown functionality used by all abilities.

IAbility

An ability is something that influences the game environment (level) in some way.

Dash

The ability for "dashing" forwards with a high speed in the direction in which the user is moving.

Reflect

The ability used to reflect projectiles. Reflected projectiles can kill enemies.

Shockwave

The ability used to push all nearby entities (excluding projectiles and obstacles) away.

Shock

Abstract helper ability for shooting different kinds of projectiles.

ShootBullet

The ability for shooting bullets.

ShootMissile

The ability used to shoot missiles.

Audio (package)

AudioHandler

Class for handling music and sound effects.

IAudioHandler

An abstraction of an audio handler.

behavior (package)

ability (package)

AbilityBehavior

Abstract implementation of an IAbilityBehavior. Simplifies the creation of new ability behaviors.

IAbilityBehavior

Ability behaviors define how an entity will use its abilities.

SingleAbilityBehavior

Simple ability behavior implementation. Only one ability is used.

movement (package)

FleeingBehavior

Movement behavior for entities that try to keep a certain distance to the target entity.

IMovementBehavior

An abstraction for defining how a movable entity moves.

SeekingBehavior

Movement behavior for entities that move straight towards the target entity.

IBehavior

Conceptual marker interface. Marks anything considered to be an ability.

entity (package)

enemy (package)

Enemy

A hostile entity that typically targets the player.

IEnemy

An abstraction of a hostile entity on the map.

movable (package)

ILiving

Abstraction for an entity that is "alive" and hence can be killed.

IMovable

Abstraction for an object which can move using physics-based on acceleration, velocity, and position.

LivingEntity

Abstract class for living entities that might be (temporarily or permanently) invulnerable.

MovableEntity

Abstract implementation of an IMovable entity. Lots of game physics is implemented here.

obstacle (package)

IObstacle

Interface for non-hostile, blocking entities that can be used for cover.

MovingWall

A wall that moves back and forwards between a start and end position.

Spikes

Obstacles that are non-hostile but can do collision damage.

Wall

A basic obstacle that blocks entity movements.

player (package)

IPlayer

Abstraction for defining a user-controlled player entity.

Player

Implementation of IPlayer. This is the class that the player controls when playing the game.

projectile (package)

Bullet

A projectile that moves in a single direction.

IProjectile

Abstraction for projectiles which can hurt the player and be reflected using.

Missile

A projectile that steers towards its target, typically the player.

Projectile

Abstract projectile implementation.

Entity

Abstract entity implementation that simplifies the creation of new entities.

IEntity

Abstraction for all entities in the game. An entity might be the player, an enemy, a projectile, or an obstacle.

IStrength

Any object with a strength (damage/hit points).

level (package)

ILevel

Abstraction for a class representing a single level/stage of the game.

Level

Level implementation, used by the game.

score (package)

HighscoreHandler

Implementation of an IHighscoreHandler. Implements functionality for writing and reading scores to/from a file.

IHighscoreHandler

Writes and reads high scores to/from a file.

shape2d (package)

Circle

Simple circle shape.

ICircle

Circle abstraction.

IRectangle

Rectangle abstraction.

IShape2D

Interface for simple shapes.

ITriangle

Triangle abstraction.

Rectangle

Rectangle shape.

Triangle

Triangle shape.

Game

Root model class. Implementation of IGame. Used to run the actual gameplay.

IGame

Interface the root model class. Defines functions that influence the gameplay, typically used by controllers.

IPositionable

Interface for any object with a position.

IUpdatable

Interface for any object which should be updated each frame.

Services

EntityFactory

Responsible for creating Entities of different sorts using methods with only a few inputs.

LevelLoader

Loads levels according to the information stored in JSON-files. Acts as an iterator of levels for the Game.

ILevelLoader

An interface used by LevelLoader. Implements Iterator to allow for iteration over ILevel objects when changing levels.

Util

Shapes

Has methods for working with shapes. Implements the collision checking algorithm used by the rest of the game for detecting and handling collisions between entities.

Timer

Used for keeping track of how much time has been spent on a level.

Utils

Methods for manipulating vectors, and some additional helper methods.

view (package)

IShapeVisitor

Interface for implementing visitor patterns for shapes.

IVisitableShape

Interface for implementing visitor patterns for shapes.

ViewResourceLoader

Loads view resources and define view-specific values.

pages (package)

abilityBar (package)

AbilityBar

Holds player ability cooldown indicators.

AbilityHolder

Holds a single ability cooldown indicator.

canvas (package)

GameCanvas

Canvas class is used to draw entities and effects on the screen.

gameState (package)

GameStatePanel

Panel for displaying information about a particular game state.

level (package)

LevelPanel

Panel for displaying acquired levels

menu (package)

buttons (package)

ExitButton

Button for exiting the game.

LevelButton

Button for changing level.

ResponsiveButton

A button that changes size depending on screen size.

ScoreButton

Button for viewing score.

StartButton

Button for starting the game.

StartMenu

The menu that is displayed at the start of the game, which provides the user with a set of options, for example, starts the game, selecting a level, etc...

score (package)

HighscorePanel

Displays player score for each level.

IScorePanel

Interface for panels displaying the current player score.

ScorePanel

A panel that displays the current player score.

MainWindow

The main window of the game. Sets up a window and various UI elements.

renderer (package)

> > **IRenderer**
>
> Abstraction for drawing entities to a canvas.

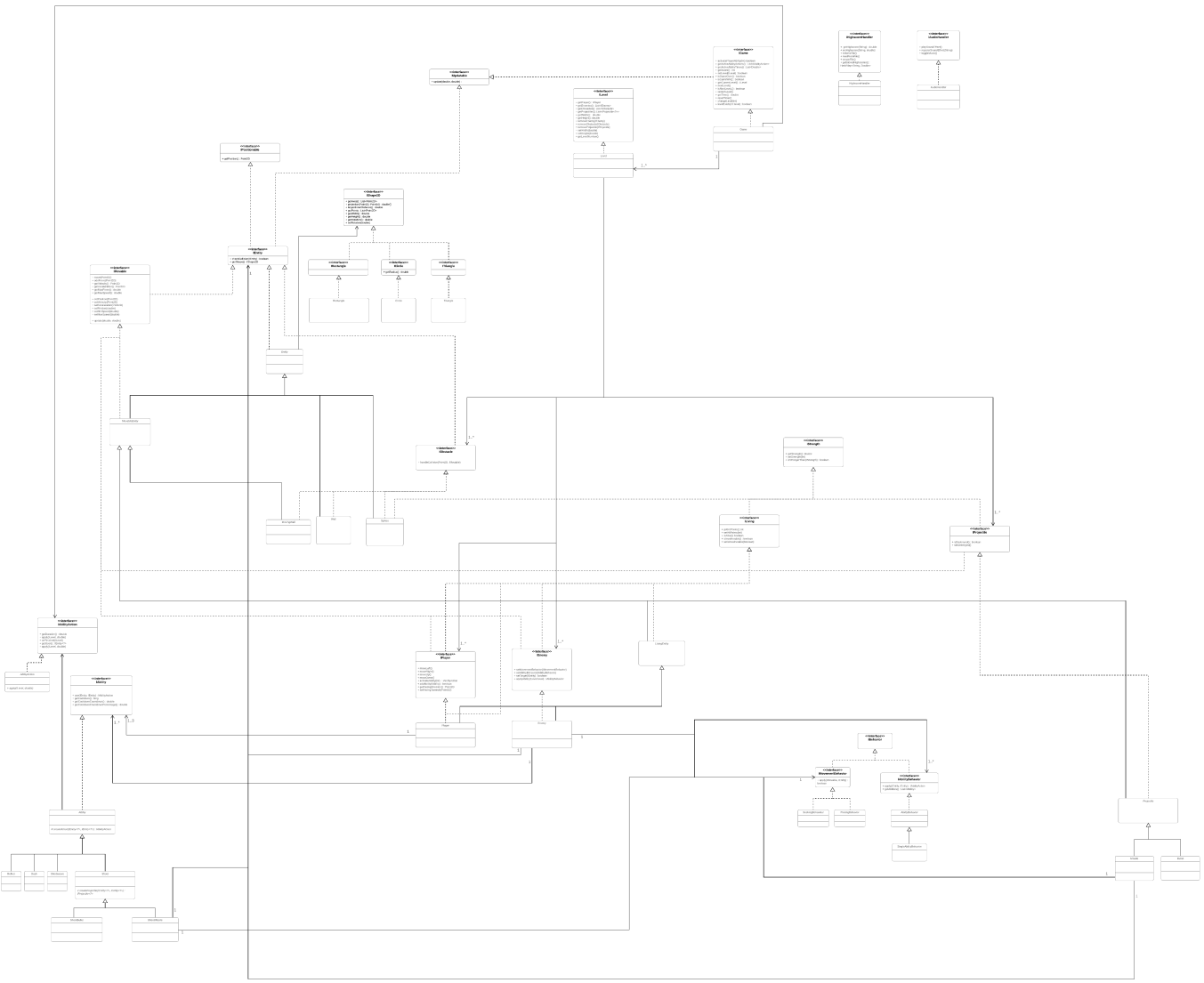
> > **Renderer**
>
> View used for rendering gameplay to a GameCanvas.

> > **RenderUtils**
>
> A layer of abstraction between view and JavaFX. Used to facilitate drawing to the screen.

> **App**
>
> Root class that launches the application.

3.2 Design Model

UML of design model



SDD

System Design Document for Pointy

Authors: Anton Hildingsson, Erik Magnusson, Joachim Ørfeldt Pedersen, Mattias Oom, Simon Genne

Version: 1.0

Date: 2020-10-23

1. Introduction

Jerk Evert is a topdown 2D game. The player is a simple geometrical shape that navigates a hostile, equally geometrical, world. In this world, the player is attacked by various enemies that shoot different kinds of projectiles at the player. The player itself has no weapon, but instead a set of abilities which (with some creativity) can be used to defeat the enemies. A few of these abilities are "reflection" (reflecting enemy projectiles), "shockwave" (pushing enemies away) and "dash" (making the player invulnerable and very fast for a short period).

The map contains different neutral elements, such as walls and moving walls, which can trap the player, but also be used as cover.

The goal of the player is to defeat all enemies. The game contains multiple levels, and by defeating one level, the player can progress to the next.

1.1. Definitions, acronyms, abbreviations

- 2D - Two dimensional
- Topdown game - A game that is viewed from above.
- Projectile - Object fired by some entity towards some other entity. Can cause harm to the player or enemies.
- Obstacle - Neutral object placed on the level. Some obstacles may hinder the movement of the player, enemies, and projectiles in the game.
- Ability - Can be used by the player or enemies to impact the flow of the game in some way. This can, for example, be by adding a projectile to the game, or by directly impacting the surrounding entities in some way.
- Player - Entity controlled by the user of the Game.
- Enemies - Opponents of the player. Will use their abilities to try to defeat the player.

2. System architecture

The general architecture of the application is rather simple. No external servers or databases used -- the game is all run locally on the machine of the user.

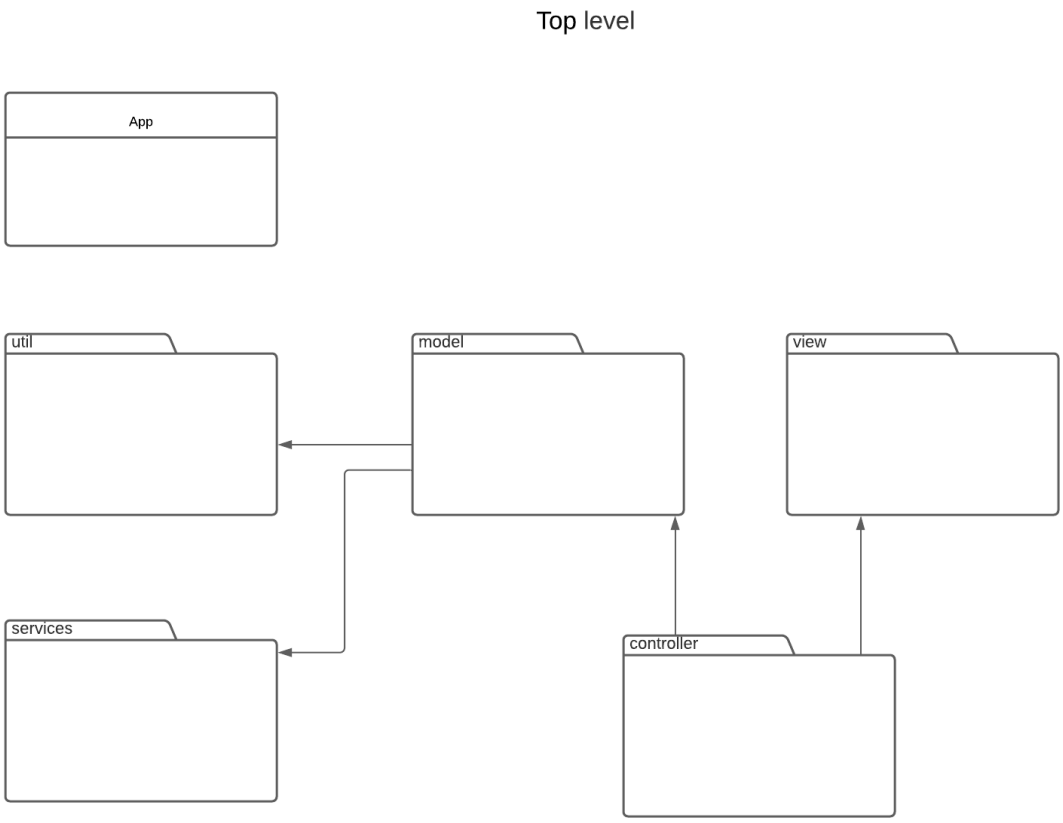
OpenJFX is used for the graphical end of the game, reading keyboard input, and handling sound. OpenJFX also manages the runnable application itself, which means a JavaFX `Application` class is created to launch the game.

Persistent data storage is all handled locally by an external JSON-parser. More can be read below (4. Persistent Data Storage).

When the application starts, the JavaFX Application loads a game window controller and initializes the `MainWindow` (the container for all graphical elements). This game window controller then creates a new `Game` object, a game loop, and launches the game. At this point, the first level is read from disk. When the player completes a level, the next level is read from disk in the same way.

The player is prompted by a menu that controls the level settings, the starting and stopping of the game itself, and displaying the score and other progress indicators. When the player starts the game, it will run until they stop it themselves, or until the game is finished.

3. System Design



The controller package interacts with the view by letting `GameWindowController` store a `Renderer` object that can be used to draw to the screen. It also has an `IGame` attribute, that will refer to the instance of `Game` used to run the game, through which it can interact with the gameplay. `GameWindowController` also creates a game loop, in which the renderer will be used to draw the current state of the game to the screen. In the loop, a call will be made to the model telling it to update its state.

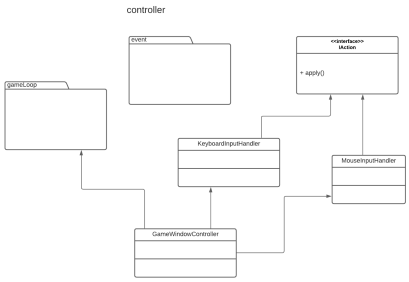
Both the `Model` and `View` packages make use of the functions implemented in the util package to affect vectors.

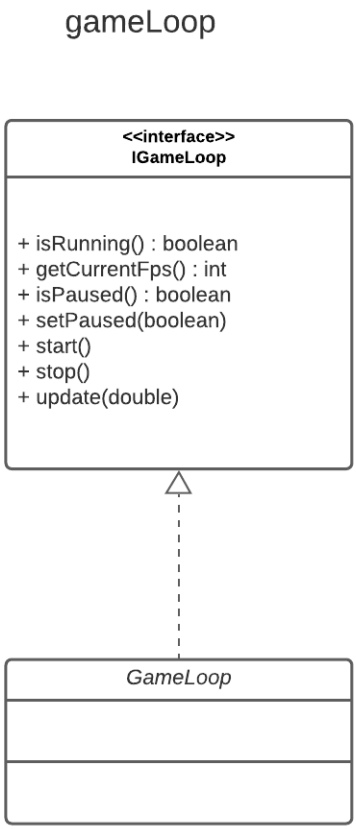
As of now, the MVC implementation is not typical. The controller (in this case, the `GameWindowController`) has access to both the view and the model. The relationship between the view and the model is limited. Most rendering is done by letting the controller pass part of the model to the renderer as the argument in a method call. However, the `GameWindowController` also registers the renderer as a `AbilityActionEventListener` for `IAbilityActionEvents` which are sent by `Game` when any entity activates an ability. This is used by the renderer to know when and how to draw certain visual effects.

Being that the view doesn't interact with the model in any way, we believe that we have managed to achieve a version of MVC that is better than the typical one since we reach a higher level of decoupling than we otherwise would have.

The `GameWindowController` also links `Game` with a `KeyboardInputController` which applies different actions depending on which keyboard keys are pressed, and a `MouseInputController` which does the same, but for mouse actions.

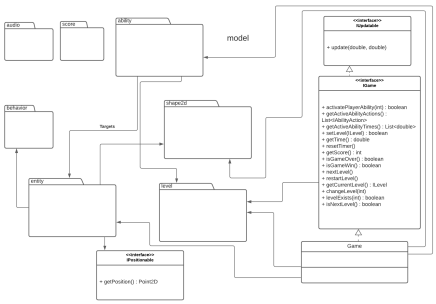
Here follows a set of diagrams over all our packages. We have decided to leave the fields containing lists of objects in the package diagrams, since the package diagrams otherwise cannot show the relationship between packages. However, we have left out these fields in the design model. Instead, these fields are represented by multiplicities.

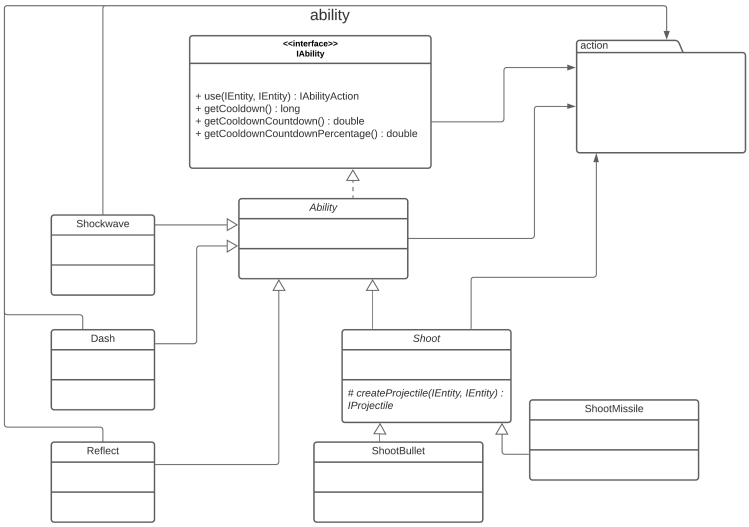




GameWindowController creates a game loop, during which, it will use the Keyboard- and MouseInputHandler to gather input from the user and apply these to the model.

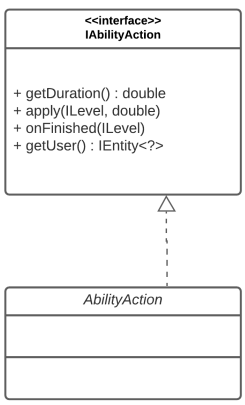
GameLoop implements the template of the game loop that will be used by GameWindowController.

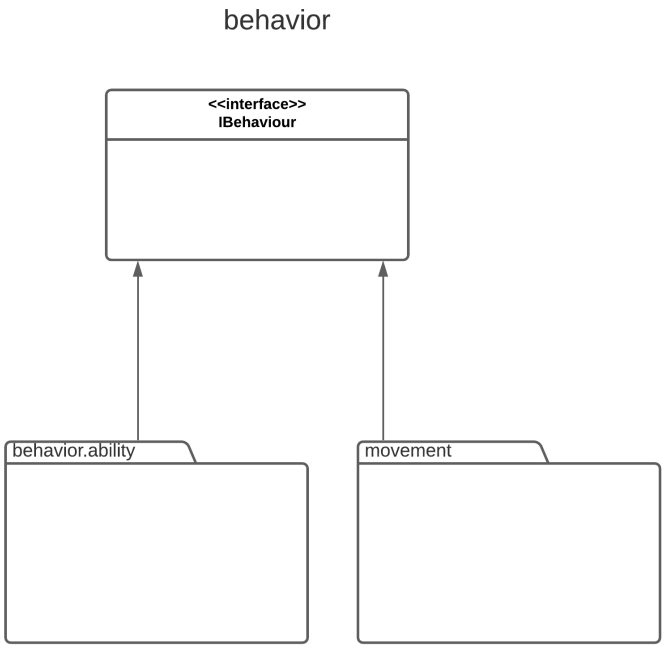




Game is what connects the different parts of the model. It will update the state of all the entities during the gameplay, according to interactions between them and inputs from the outside.

Abilities can be used by the player and enemies during gameplay to impact other entities.

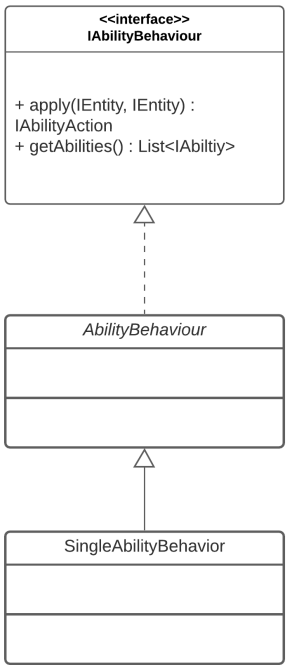




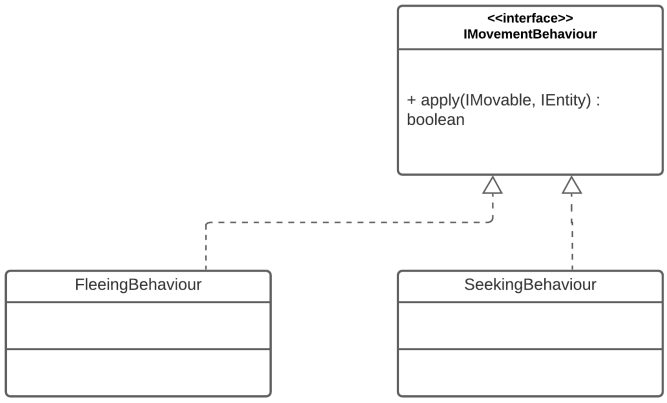
AbilityActions are created by Abilites to apply their desired affects onto the game.

Behaviours control the actions of enemies.

behavior.ability

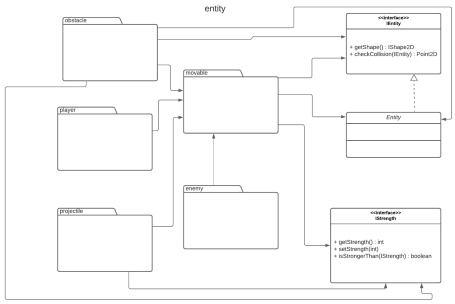


movement

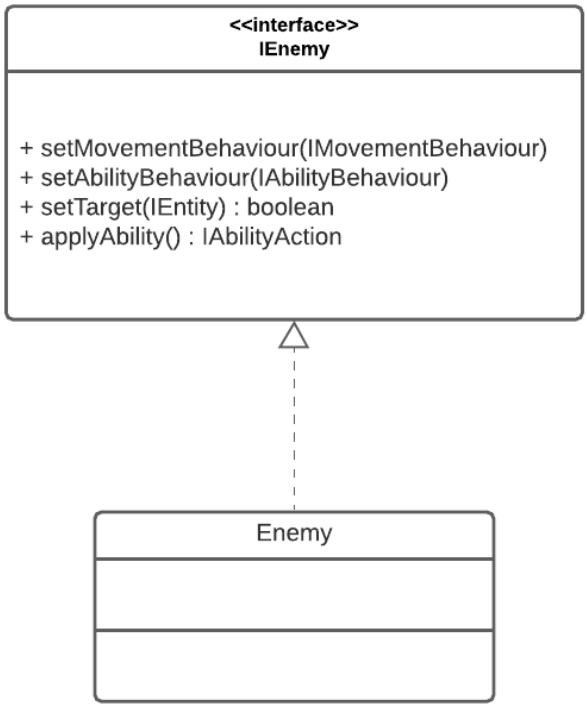


AbilityBehaviours control how an enemy uses its abilities.

MovementBehaviours control the movement of an enemy.

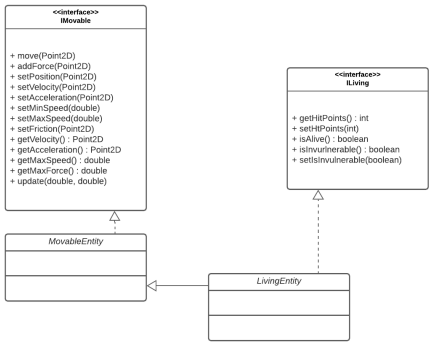


enemy

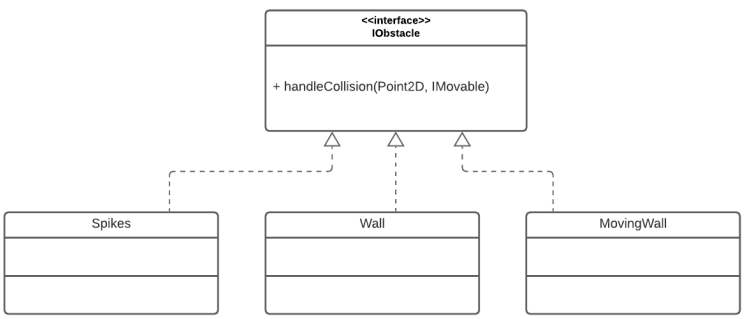


Entities represent all active objects in the game, such as players, enemies, obstacles, projectiles, and so on.	Enemies are the opponents of the player. Acts according to its behaviours. Different enemy types are achieved through different combinations of behaviours.

movable

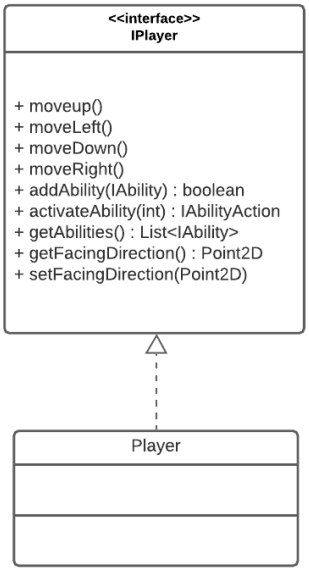


obstacle

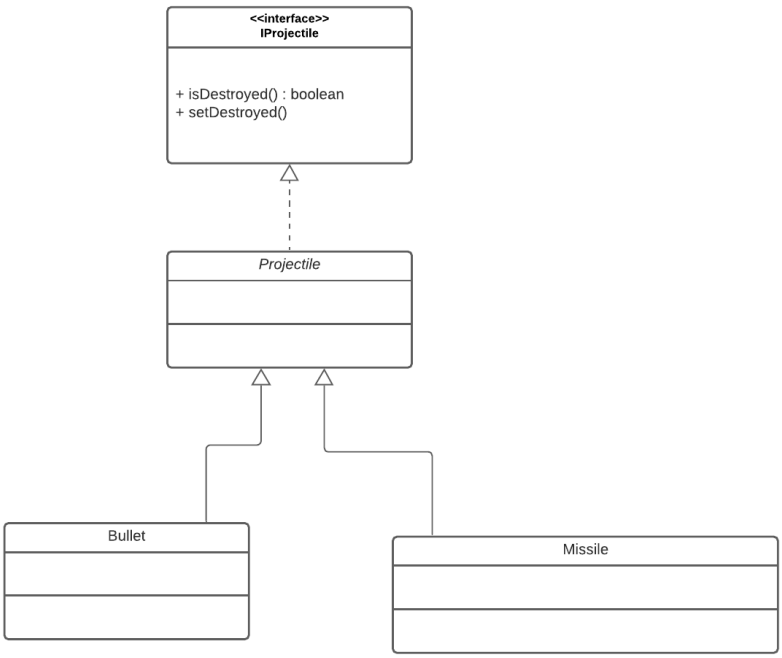


MovableEntity represents something that can move. Implements the movement functionality for all movable entities in the game.	Neutral elements that can hinder the movement of other entities and/or cause damage to them.

player



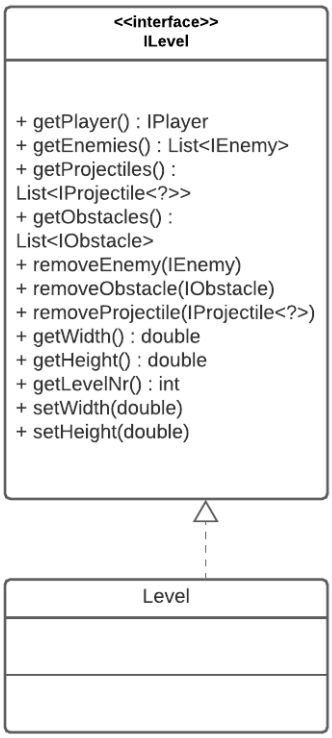
projectile



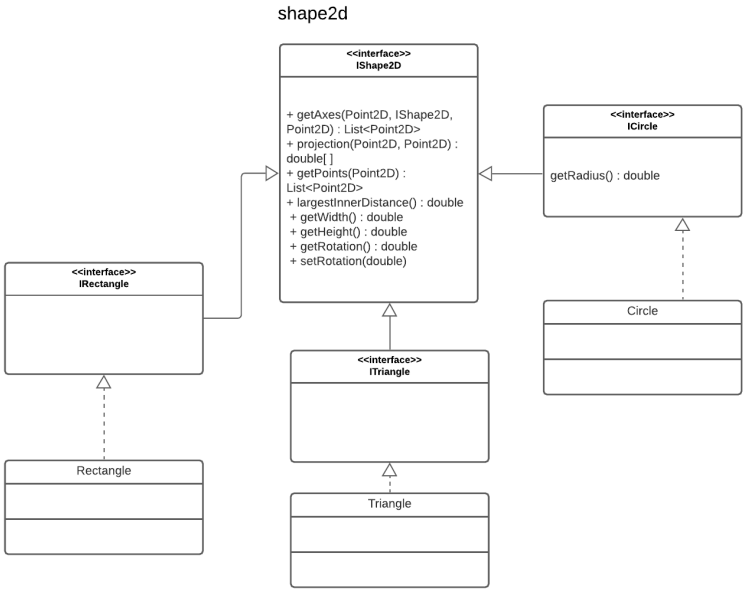
Player represents the entity controlled by the player of the game.

Projectiles can be fired by some enemies and cause damage to certain entities. Bullets will get a velocity when created, which will remain until they are destroyed. Missiles will change their velocities according to the movement of their targets.

model.level

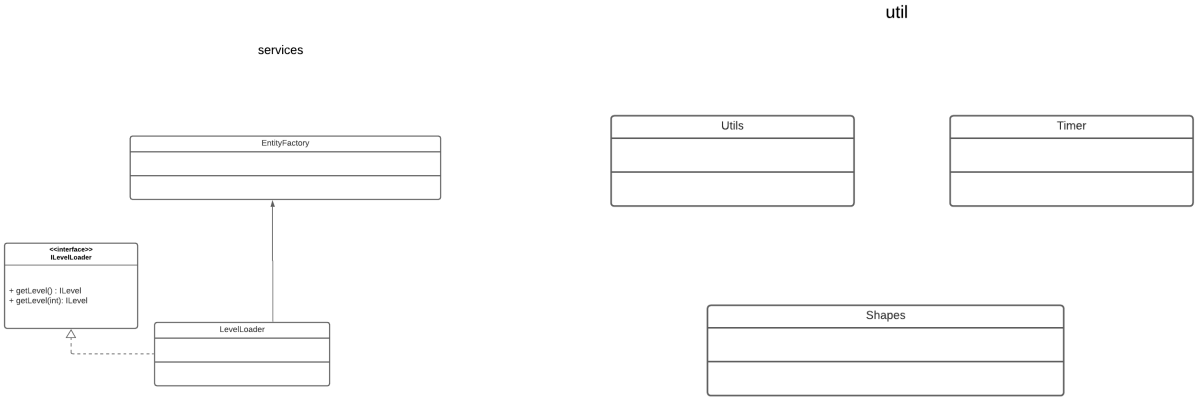


shape2d



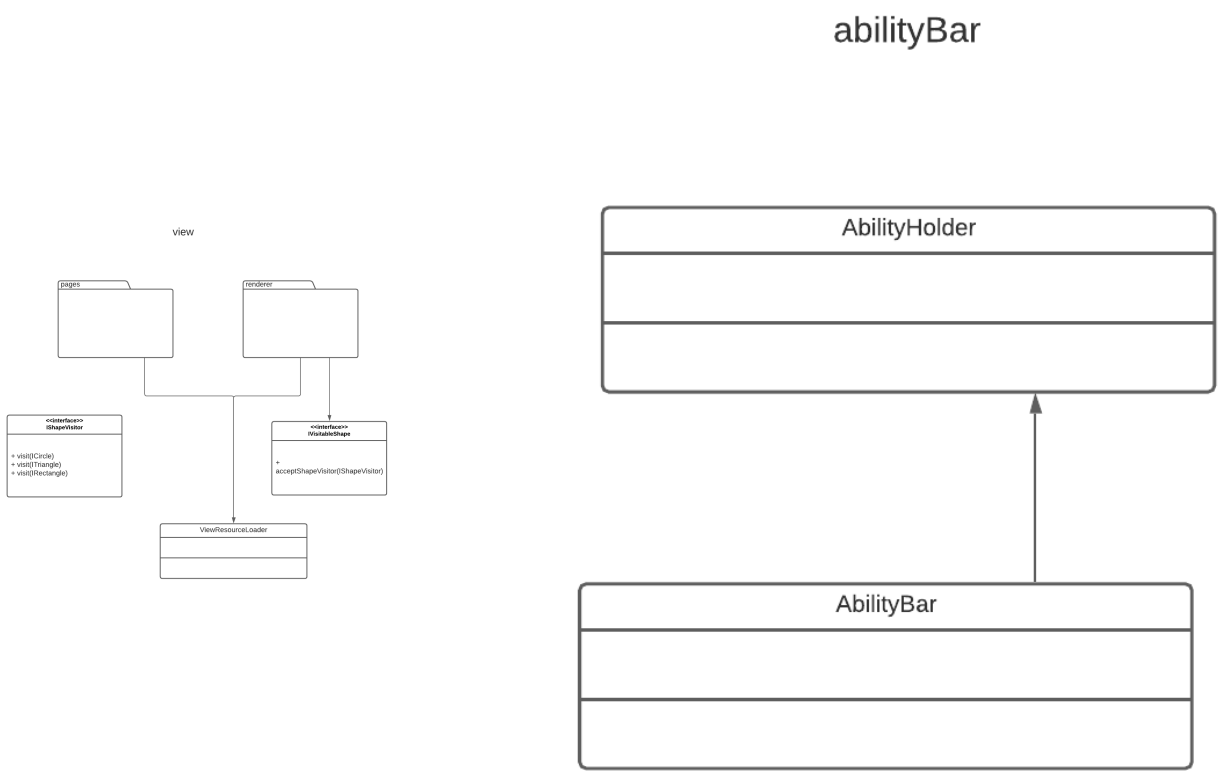
A Level stores the contents of a playable level, together with information about width and height of the map.

Shape2D represents the shape of an entity. It stores methods relevant for rendering and collision checking.



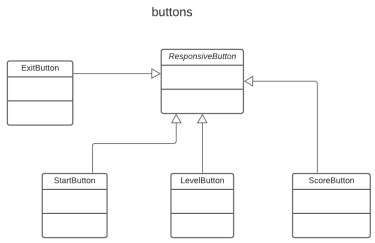
LevelLoader creates levels from JSON-files that specify the contents of the level. EntityFactory creates entities of different sorts.

Utils contains helper methods for handling vectors. Shapes contains methods for manipulating shapes.

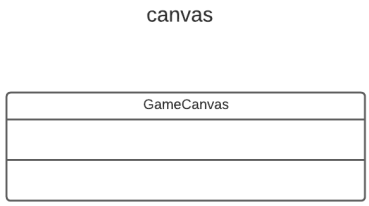


Renderer uses RenderUtils to draw entities to the screen.

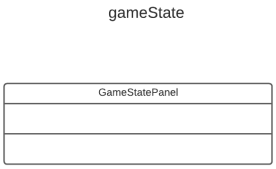
AbilityHolder contains status of ability and AbilityBar contains multiple AbilityHolder, and displays the status of the ability.



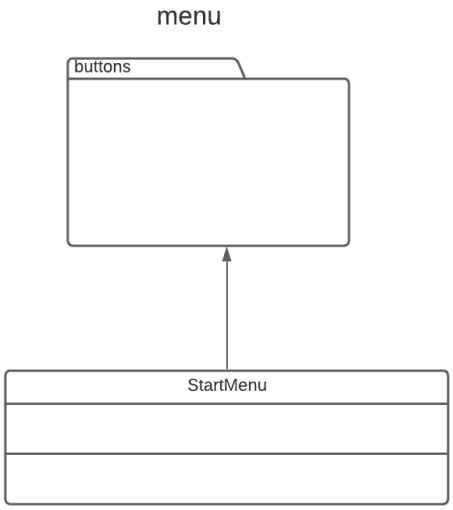
Buttons for the start menu



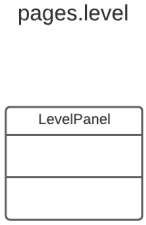
GameCanvas is used to draw entities and effects on screen.

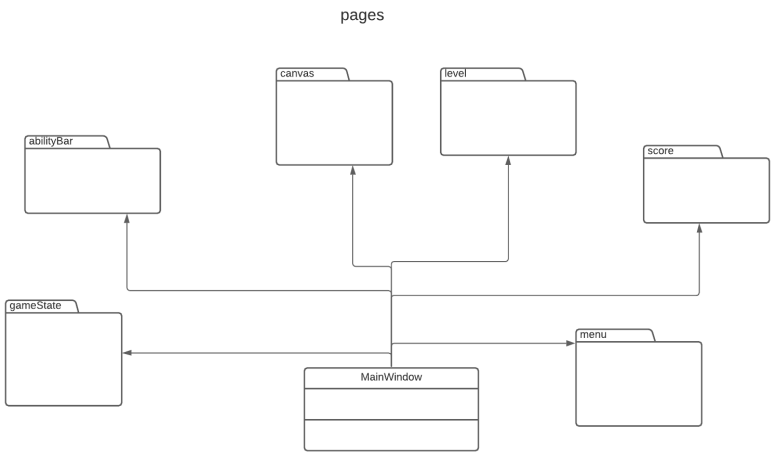


Gamestate displays the current state of the game.

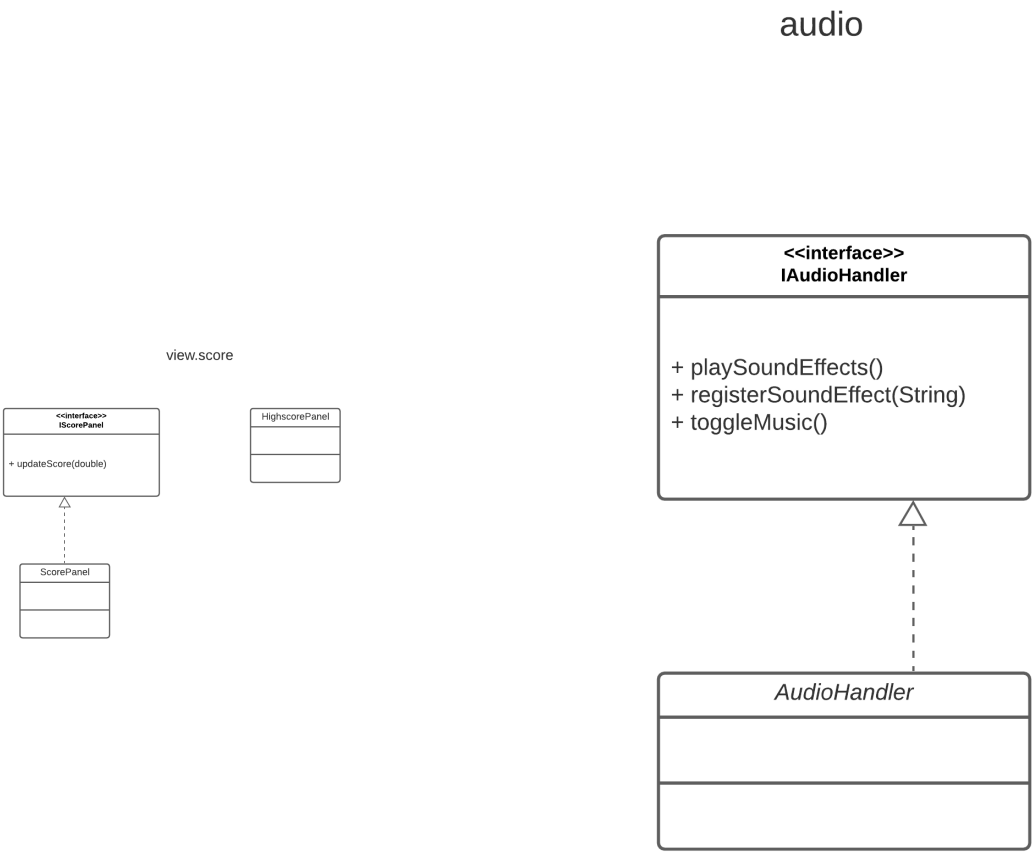


Start menu contains buttons and displays them on the screen.



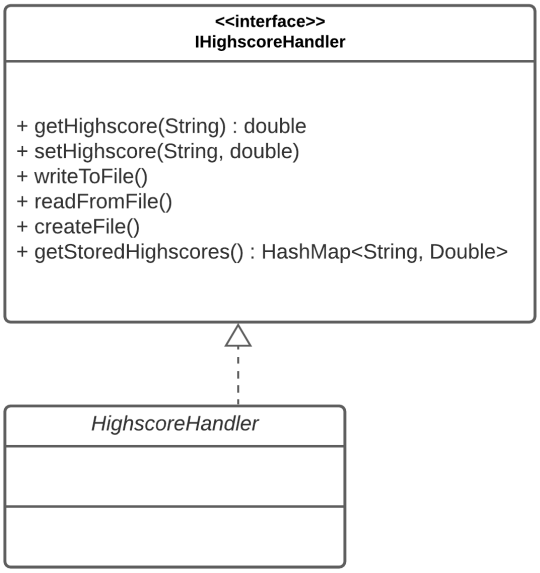


Levelpanel displays the acquired levels	Contains all graphical components.

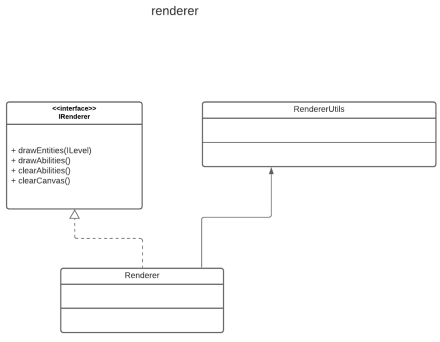


ScorePanel is used to display the time during a level, and HighscorePanel is used to display the Highscore of every completed level	Package for handling music and sound effects

model.score

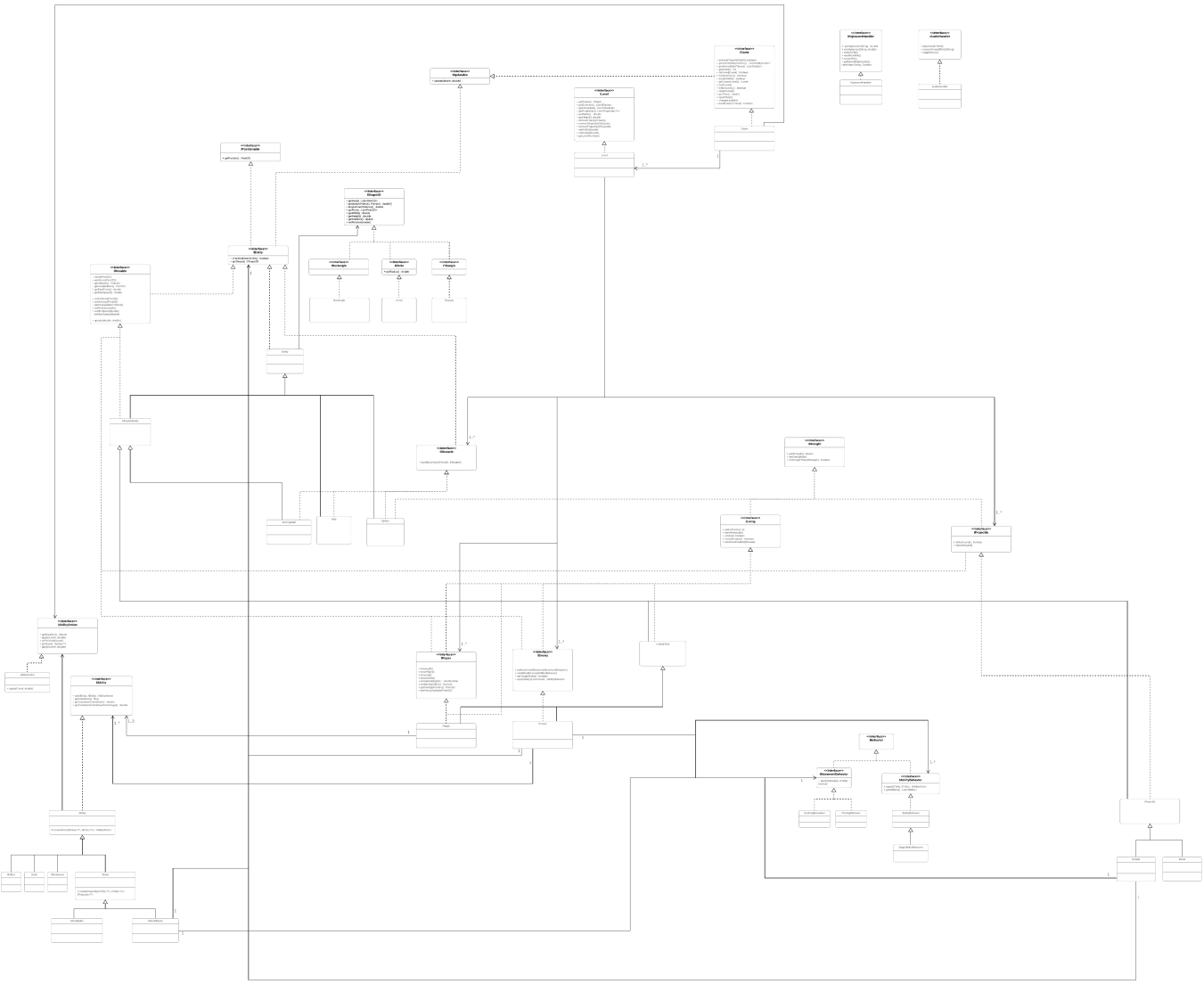


Package for handling ability related events, used to notify view when abilities are activated	Reads and writes score to disk and used to display score when playing the game.

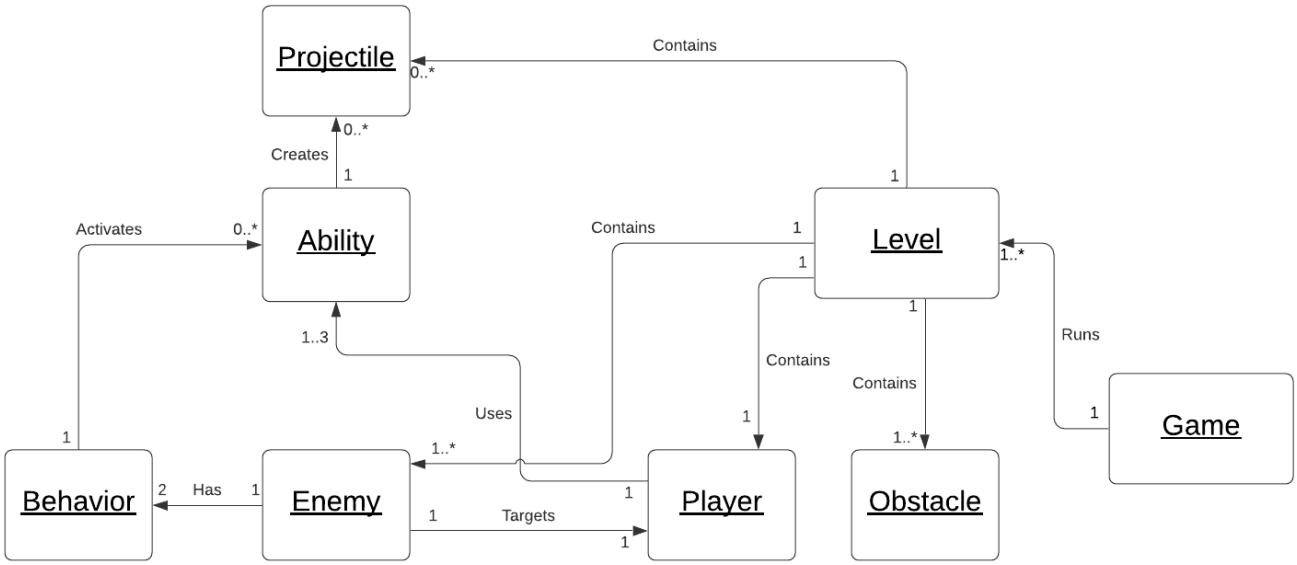


Provides functionality for drawing to the canvas

Design model:



Domain model:



3.1 Relation between domain model and design model

In the domain model, The `Game` class is said to run the `Level` which contains a player, enemies, obstacles, and projectiles. This is reflected in the design model, where `Game` has a reference to an `ILevel` (currentLevel) and

a list of `ILevels` (levels). `Level` holds references to the enemies, obstacles, player, and projectiles, that are to be shown while on the level that is represented by that object. `Game` will, during gameplay, access these and update them according to the state of the game and input from the user.

The domain model shows `Enemy` to have two behaviours. In the design model, this is the case since `Enemy` has a reference to an `IAbilityBehaviour` and an `IMovementBehaviour`. These will dictate what actions are carried out by the enemy.

Both the `Player` and `Behaviour` in the domain model have references to `Ability`. In the design model, `Player` has a reference to one to three `IAbilties`. These abilities will be used by the player during the gameplay to affect the environment/state of the player in some way. The multiplicity of behaviours related to `Ability` is 0..* in the domain model. In the design model, some behaviours will have no knowledge of abilities (movementBehaviours), while some will be able to hold many (abilityBehaviours).

The `Ability` in the domain model creates 0..* projectiles. In the design model, some concrete ability classes have references to projectiles. These abilities are supposed to create projectiles and add them to the level. Other abilities have no knowledge of projectiles at all.

3.2 Implemented design patterns

- MVC (Model View Controller) for separating game logic, user input, and graphical interface.
- Factory (method) pattern for simplifying the creation of game entities such as players and enemies
- Command pattern, which is used for executing actions when a key is pressed.
- Composite pattern, allowing players and enemies to have different abilities and behaviours. The construction of these entities is simplified using the factory pattern.
- Template method pattern, letting Ability implement a method that is dependent on an abstract method implemented by subclasses.
- Visitor pattern, shapes accept a visitor to do specific action for different shapes
- Observer pattern, the view listens for ability action events sent by the game. This is used to indicate when to draw certain effects to the screen.
- [NOTE] Singleton pattern is an option for KeyboardInputHandler (previously were) and MouseInputHandler, but this results in harder to test code and less readable code in general since we could reference the instance from anywhere in the codebase. Hence we no longer use the singleton pattern.

4. Persistent data management

The application makes use of JSON to handle level data. The level files contain JSON objects pertaining to the information of the level and its entities, i.e. their type (player, enemy, obstacle), variants (e.g. type of enemy) as well as instance variables not handled by the factory. Levels are loaded through the class `LevelLoader` which parses the JSON file corresponding to a certain level ID, creates an object of type `Level`, and returns this object to be used by the `Game` class. Each level is contained within a separate file and is only loaded when needed to save memory resources. The parsing is done using the GSON library.

Top scores for each level are saved in a text file which is handled by class `HighscoreHandler`.

5. Quality

The application is tested using unit tests with the framework JUnit. These tests can be found under project/src/main/test.

The overall code coverage of the application is 45%. The coverage of the model, however, is around 82%.

Continuous Integration is done using Travis to automatically run tests when doing pull requests to ensure nothing merged to master causes tests to fail.

Link to Travis for our project: https://travis-ci.com/github/feldtsen/pointy-dit212

The game has also been thoroughly game tested by playing the different levels and manually testing the functionality described by the different user stories.

6. References

- JavaFX - https://openjfx.io/
- JUnit - https://junit.org/
- Maven - https://maven.apache.org/
- Travis - https://travis-ci.com

Peer Review

General remarks

The code was often simple to read but took some time to understand due to a lack of documentation and a comprehensive design model. However, the code had some issues, which we'd like to address here. First of all, the model (`QuizModel`) lacks some functionality, which is instead placed in the class `StandardQuizViewModel`. For example, the methods `answerQuestion`, `createAlternativeList` and fields such as `totalQuestions`, `correctAnswers` and `questionProvider` could be contained within the model with the appropriate methods added to the model interface (`IQuestionHandler`). This would increase the separation between the view and the model. This separation is currently fairly limited, or at least unclear.

Sometimes, the design patterns used, or how they are implemented, strike us as contrived. For example, all your model observers only implement one method (`quizFinished`), and don't handle any events. Any model observer could therefore only listen to one type of action, which is not extensible. This breaks both DIP and OCP. Polling could easily be used, especially since the `StandardQuizViewModel` subscribes as an observer to one of its own fields (`questionHandler`).

The use of the iterator pattern is questionable. All IQuestions have an iterator which iterates over the question alternatives, however this iterator is most often later converted to a list, for example in `createAlternativeList` in `StandardQuizViewModel` (in this case, this list is also later converted to a list of tuples, which is unnecessarily complex). We found the same issue in `QuizModel`. Often, the iterator pattern does not aid your model, it just makes things more complicated. Consider if other data structures, for example sets or maps are more suitable. (You don't have to implement a list iterator on your own, since all java lists have an `iterator` method which returns an iterator for that list.)

Factory classes are often used, which is good for extensibility and modularity, however most of your factories only implement a single method or perform fairly simple operations. For example, `createStandardModel` in

`ModelFactory` could just be a constructor call (if `QuizModel` takes the iterator of questions in its constructor instead of using the `insertQuestions` method).

There are some other small issues sprinkled around the code too. For example, `setBaseQuestion` in `ScrambledQuestion` is used instead of a constructor call. This method does nothing if it has been called before, which is unintuitive, especially when calling this functionality easily could be disallowed by using a constructor call instead.

The name of `QuestionsFromFile` doesn't actually read from a file. The implementation for generating random math questions is also a bit strange. Wrong alternatives are generated by adding the two numbers together, and then adding a random value. Why not just randomly generate a wrong alternative (and retry if it ends up being correct, by chance)?

Your tuples should be more descriptive. Instead of creating a general tuple class, create a more descriptive class (for example, “Alternatives”) with fields actually describing the intended contents of the class.

The code uses switch statements at several places which could make it hard to extend and maintain. There is also duplicated code at some places, for example the constructor of `StandardQuizModel` and `changeQuestion`, as well as `getQuestion` and `nextQuestion`.

Structure and Documentation

The System Design in the SDD doesn’t seem to correctly reflect the project. The diagram shows “QuizModel'' being a package but in the project it’s called “QuizPackage”. There is no designmodel or other detailed UML-diagrams in the SDD or RAD which creates difficulties in overlooking and understanding the project. It is also hard to see how the domain model relates to the code. “User” exists in the domain model but doesn't seem to have any corresponding component in the code.

The project could make use of a more solid package structure, for example in activities, modalFragment and RecyclerViewAdapter.

We generally find that the code is lacking in comments, at least to make it understandable for anyone not already familiar with the project. Several classes do not contain descriptions of what they are or what they do. Likewise, comments pertaining to methods are quite often vague and only useful for someone who’s already well acquainted with the code.

Style and conventions

The style and naming used in the code is varying and often doesn’t follow convention. Here are a few small notes: Many interface methods are unnecessarily prefaced with “public”, as seen in the `IViewModel`. The modal field in `MainActivity` is not private when it should be. Some methods suffer from strange formatting, such as `onCreate` in `SettingsActivity`. And the `CountDown` method in `QuizActivity` is capitalized when it shouldn’t be. The opposite could be said for your package names: java packages should not be capitalized.

Many variables are also poorly named. The name of “tupleList” should reflect the contents of the list. “observerList” could just be “observers”, “questionStack” could just be “questions”, and so on.

Testing

You have a good amount of tests, however some are odd. For example, `QuestionFactoryTest` only tests if a returned object has the "correct" type. There's also an example-test left in the code. Final thoughts You have a good idea, and what you describe in your SDD seems good. The described structure is good and makes sure you can separate model and view, even though android application enforces the use of certain classes. However, your code doesn't exactly match your descriptions or your domain model. Hopefully our suggestions will be of good use, and help you build an even better application.

Good luck!