Software Design Description of Denty

1 Introduction

1.1 Design goals

The design should provide for the possibility of easily extending the game with new levels, enemies and blocks. The design also has to be testable to verify that different parts of the program works. Furthermore the future directions shall not cause the need of major refactoring when they are implemented.

1.2 Definitions, acronyms and abbreviations

- **Block**, a solid square which is either static and attached to the level, or dynamic as a part of the physics environment.
- **Denty**, the main character controlled with the keyboard and represented as a sprite in the game view
- **Environment**, the set of all objects that compose a playable level. Interaction with the environment means interaction with any object within the above mentioned set.

- **Enemy**, a character residing in a level, with the aim to kill Denty. Enemies can be eliminated by Denty and by the mouse player throwing blocks at it.
- **Level**, an area containing an environment that Denty is to navigate through. Has a goal in the end of the navigation, when Denty reaches this goal he clears the level and continues to the next step in the game.
- **MVC**, a way to partition an application with a GUI into distinct parts avoiding a mixture of GUI-code, application code and data spread all over
- The two players are separated by their input methods, The **"Keyboard player"** controls the main character. The **"Mouse player"** controls a block pointer that performs the placing, detaching and throwing of blocks.
- **GUI**, graphical user interface.
- **Java**, platform independent programming language.
- **JRE**, the Java Run time Environment. Additional software needed to run an Java
- application.
- **Host**, a computer where the game will run.

1.3 References

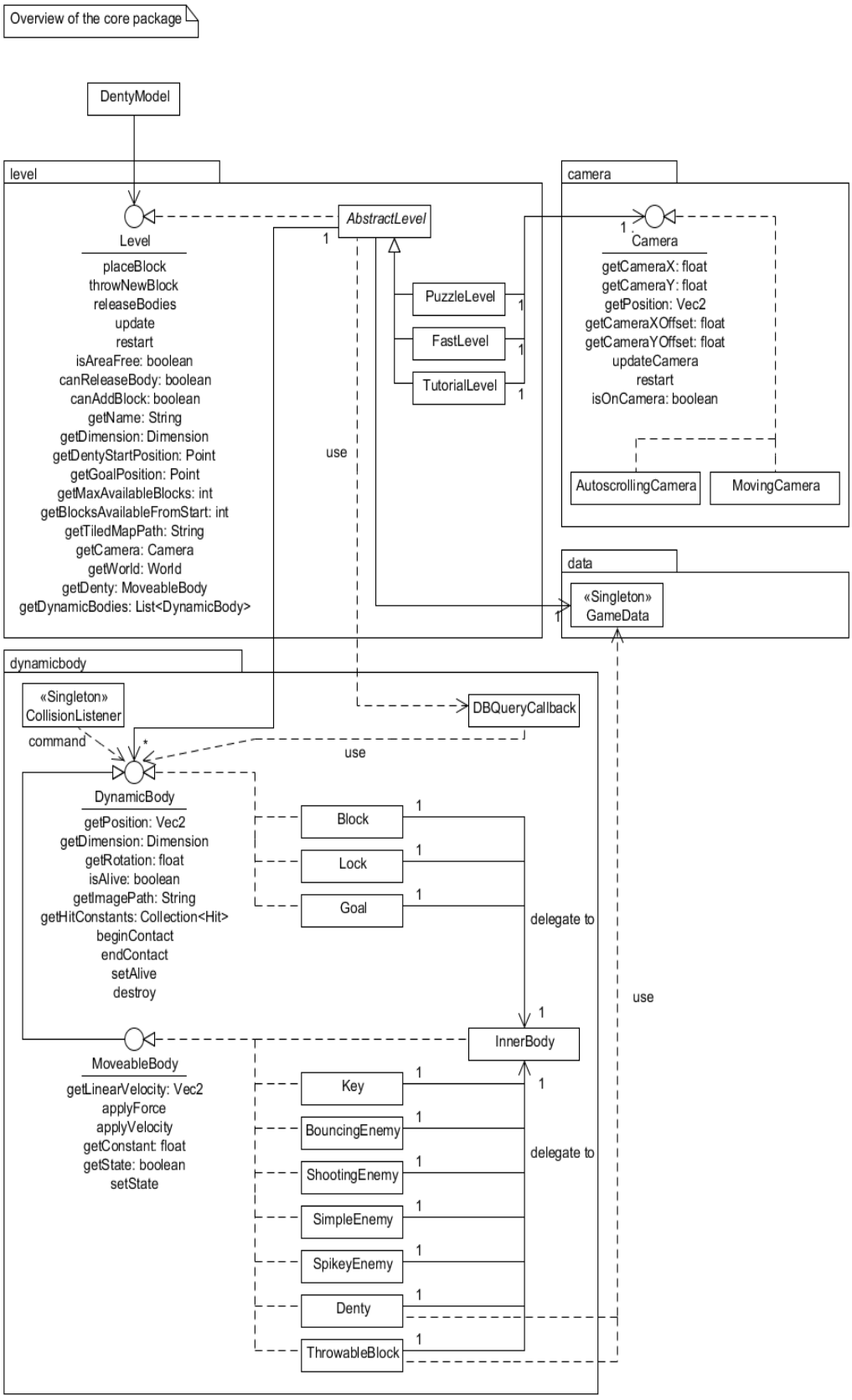
MVC, see http://en.wikipedia.org/wiki/Model-view-controller

2 Proposed system architecture

In this section we propose a high level architecture.

2.1 Overview

The application will use a MVC-structure with a model that's somewhere between fat and anemic. The InputController class registers input from both the mouse and the keyboard player and then calls the right method for each input. The class GameData is a globally reachable class with almost no logic, it manages global data displayed in the HUD. It also serves the rest of the application with a PropertyChangeSupport that view classes listens to, to know when the model classes has been updated. To make testing easier the Model, View and Controller should be as separated as possible. With a good separation it will be sufficient to test Controller and Model classes, which is very desirable when using the Slick framework in the view, since instantiating something with a Slick object in it requires starting up OpenGL. In other words, the best would be to keep the Slick objects as far away from the game logic as possible.


Overview of the core package
DentyModel
level
Level
placeBlock
throwNewBlock
releaseBodies
update
restart
isAreaFree: boolean
canReleaseBody: boolean
canAddBlock: boolean
getName: String
getDimension: Dimension
getDentyStartPosition: Point
getGoalPosition: Point
getMaxAvailableBlocks: int
getBlocksAvailableFromStart: int
getTiledMapPath: String
getCamera: Camera
getWorld: World
getDenty: MoveableBody
getDynamicBodies: List<DynamicBody>
AbstractLevel
PuzzleLevel
FastLevel
TutorialLevel
use
camera
Camera
getCameraX: float
getCameraY: float
getPosition: Vec2
getCameraXOffset: float
getCameraYOffset: float
updateCamera
restart
isOnCamera: boolean
AutoscrollingCamera
MovingCamera
data
«Singleton»
GameData
dynamicbody
«Singleton»
CollisionListener
command
DBQueryCallback
use
DynamicBody
getPosition: Vec2
getDimension: Dimension
getRotation: float
isAlive: boolean
getImagePath: String
getHitConstants: Collection<Hit>
beginContact
endContact
setAlive
destroy
Block
Lock
Goal
delegate to
InnerBody
use
MoveableBody
getLinearVelocity: Vec2
applyForce
applyVelocity
getConstant: float
getState: boolean
setState
Key
BouncingEnemy
ShootingEnemy
SimpleEnemy
SpikeyEnemy
Denty
ThrowableBlock
delegate to

2.1.1 The level

The Level-classes contains the physical representation of the game and keeps track of its bodies. Each level can have its own set of bodies and its own camera.

When a level is initiated it creates its static parts, adds DynamicBodies and starts stepping the physical simulation.

2.1.2 Bodies

The levels contain bodies. They are governed by the DynamicBody interface and its subinterface MoveableBody. The bodies can be static or moveable and have force and velocity applied to them. Everything that can be manipulated in the physical representation is a DynamicBody. This includes Denty and Enemies that move around the level, ThrowableBlocks that are thrown and Blocks that are placed in and removed from the Level. There are bodies in each level that cannot be manipulated in any way. These are not DynamicBodies and are added at the initiation of the levels and then left at the same place.

2.1.3 Controlling bodies

There are several controllers that handle the manipulation of DynamicBodies. The MoveController standardizes how they move and jump, the AutoMoveController contains the logic of DynamicBodies that move automatically (for now Enemies, but in the future there might be more automoving bodies). The CollisionListener listens for collisions in the physical world and then tells the colliding DynamicBodies about the collision. The colliding DynamicBodies will then according to DynamicConstants handle damage. RecycleController removes bodies that has fallen out of the level.

2.1.4 Collisions

All collisions are handled by the colliding DynamicBodies. When two bodies collide they are notified by the CollisionListener. The collisions are handled in two steps. Each participant is asked for Hit constants, which is a list of enums describing how that participant intends to collide with the other participant (though not knowing who or what it is). This list is passed by the CollisionListener to the other participant, which is responsible for acting according to the passed information through beginContact() and endContact() respectively. beginContact() is called at the initial touch, and endContact() when a touch expires.

2.1.5 Camera

Each Level contains an implementation of the Camera interface. The Cameras keep track of what part of the level the view should render, and which DynamicBodies that should be rendered as well as where they should be rendered. The Camera might be static, follow Denty

or move independently of other parts of the game.

2.1.6 The view

When told to render the game the view first asks the camera for its position. Then it renders the static parts of the level (with background and foreground layers.). Then for each DynamicBody the Cameras isOnCamera method is called with position and size of the DynamicBody in question, if the DynamicBody is on camera it is rendered. The last step of the rendering is to tell the HUDView to render information about lives, blocks, health points and level name.

2.2 Software decomposition

2.2.1 General

Denty is composed of three major systems: Slick Framework, JBox2D Physics engine and our game implementation. Slick is mainly used to simplify graphics rendering and to update the logic of the game at event intervals. JBox2D simulates all the physics of the game to make applying forces and velocity to bodies easier.
The Slick framework requires a few classes on our part at the highest level of the dependency hierarchy, DentyGame and DentyModel, and forces us to use frequent updates of the rendering and model instead of an event driven solution. Luckily, JBox2D works the same way.
Packages described

- denty/denty.core, contains classes that initiate the application and loads a game according to Slick
- denty.core.level, specific level classes, common physics initiation, an interface that all levels implement and an abstract class that all levels extend (to reduce duplicated code)
- denty.core.dynamicbody, all objects in the level that also has a representation in the physical world. Contains an InnerBody class that all other DynamicBodies own one of and delegates methods to.
- denty.core.camera, dictates what parts of the level and which DynamicBodies that will be drawn by the view
- denty.ctrl, controllers that manipulate dynamic bodies, take input and handles updates
- denty.constants, public constants and enums
- denty.util, public utilities including “filter by class”
- denty.view, renders graphical elements.
- denty.core.data, contains in-game data, such as hp and lives, handles listening

2.2.2 Tiers

N/A

2.2.3 Communication

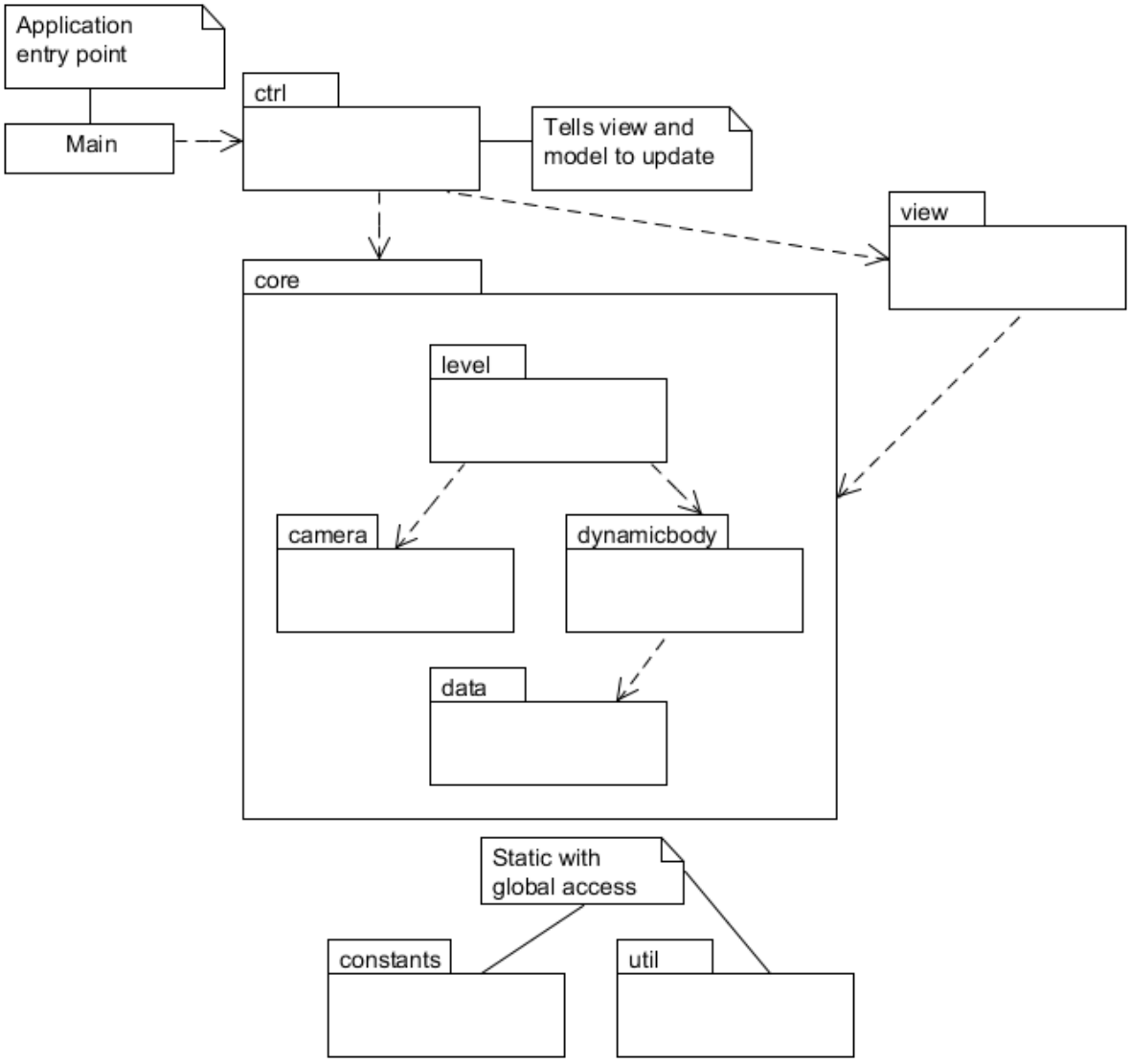
N/A

2.2.4 Decomposition into subsystems

Physics calculations are fully delegated to JBox2D.

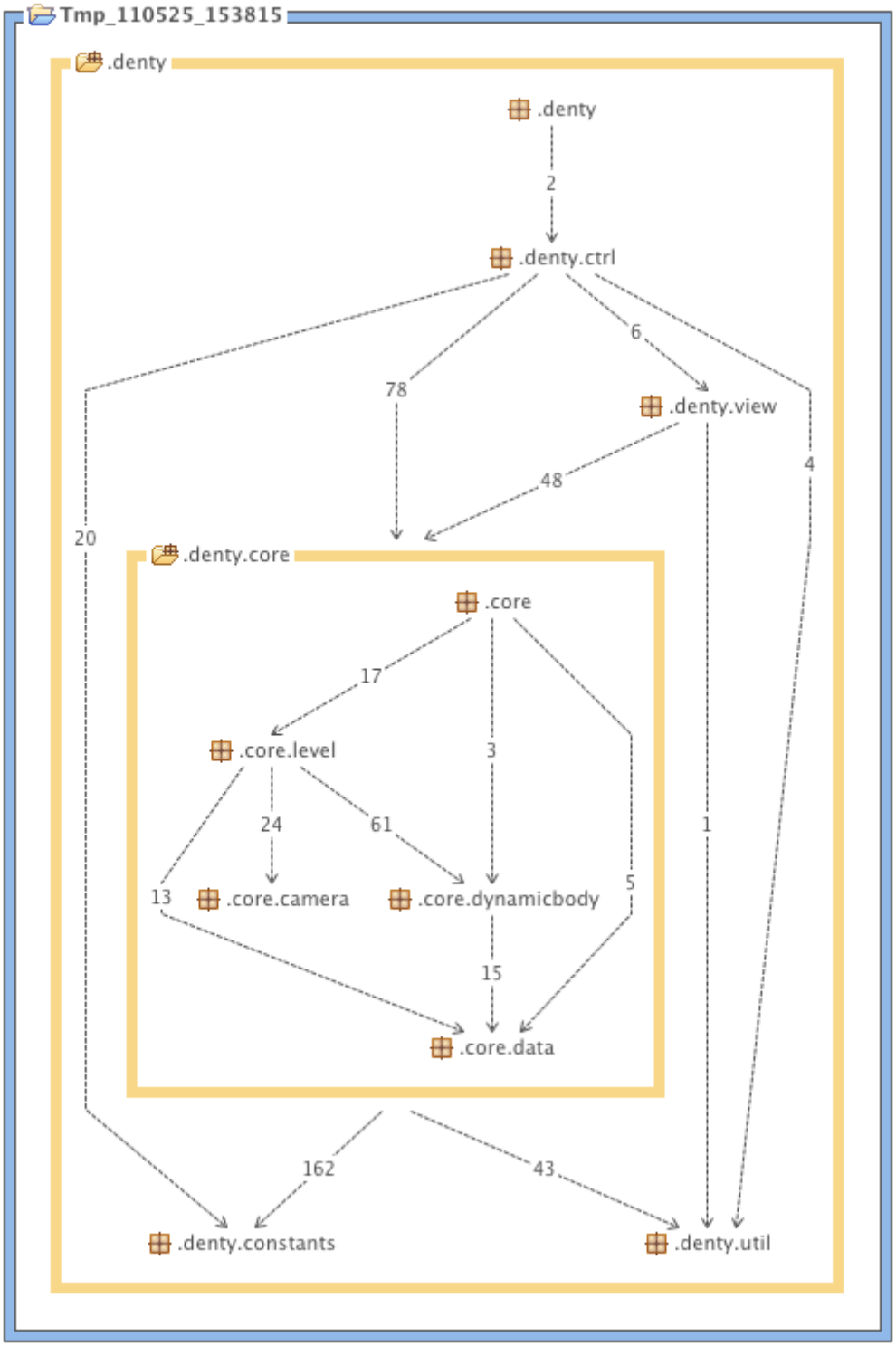
2.2.5 Layering

The layering is as indicated in Figure . Higher layers are at the top of the figure.



2.2.6 Dependency analysis

Downwards arrows indicate dependencies in the figure above.
The figure below shows the dependency analysis from STAN.



2.3 Concurrency issues

N/A, the application is singlethreaded.

2.4 Persistent data management

Tilemaps are stored in separate files and loaded with each level class. Images of Denty, enemies and blocks (i.e. different DynamicBodies) are stored in separate files and loaded by their respective classes. No other data needs to be stored.

2.5 Access control and security

N/A

2.6 Boundary conditions

N/A