15/05/2015

Emergent Architecture Design

Games Context Group 1 (a.k.a. Funky Donkey Studio)

Danilo Dumeljić	ddumeljic	4282442
Stephan Dumasy	sdumasy	4286723
Dennis van Peer	dvanpeer	4321138
Olivier Dikken	odikken	4223209
Jonathan Raes	jmraes	4300343

Abstract

This document contains our game’s architecture design.

(Note the word “currently” has many occurrences. This can be read as a synonym of “at the time of writing”)

1. Introduction

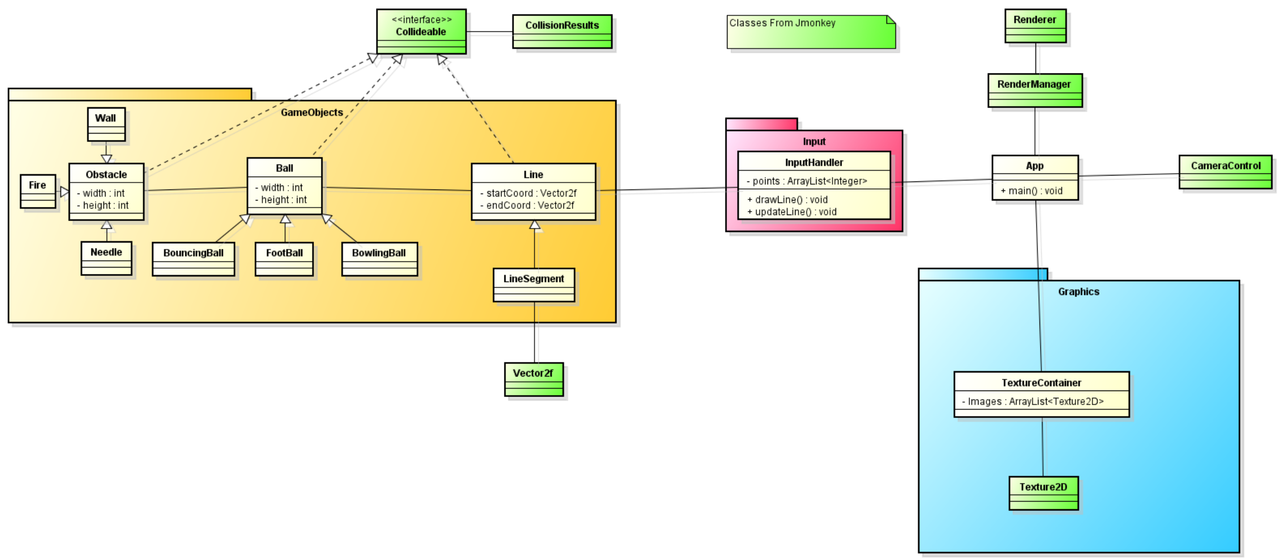
1.1. Design goals

We will try to keep system structure as simple as possible (without having a negative effect the end product), since we have a very specific setup which imposes many constraints on the possible locations where the game can be realised. Our hardware setup needs to be able to perform correctly for a real queue, without being an obstacle to the environment. Furthermore the player input via camera detection needs to be able to run smoothly in real time for the product to be playable. Our game does not require persistent data, nor does it require concurrency.

2. Software Architecture Views

2.1. Subsystem Decomposition (modules and dependencies between them)

A camera detection system gets input from the players. An algorithm detects the highest points of the foreground named control points. These control points are accessed by the input handler making use of the bridge pattern. This is done so that the camera detection system and input handler can be modified without interrupting the proper functioning of the application.



Our architecture decouples the different aspects of our application. In this way the physics, gameplay objects, GUI and camera detection modules can be worked on independently.

2.2. Software & Hardware Requirements

Player Input system

- ❖ Hardware:
  - ➢ Camera

    - Computer
- Software:
    - Camera image describes silhouettes (background - foreground)
    - Algorithm detects highest points on silhouettes and generates a dataset which will be interpreted to generate a wave

Game controller

- Hardware:
    - Computer
- Software:
    - Process player input received from the “player input system”
    - Position the object, slope, targets and obstacles
    - Check when the object reaches the target
    - Handle event object-obstacle collision
    - Keep track of combo count
    - Decide when to display powerups and when to trigger bonus levels
- Makes use of data from:
    - Player input system
    - Game physics system

Game physics system

- Hardware:
    - Computer
- Software:
    - Set gravity, friction and other physics rules
    - Compute the line / slope using the game controller data
    - Compute how the object moves along the line
- Makes use of data from:
    - Game controller

Display system

- Hardware:
    - Screen/Projector
    - Computer
- Software:
    - Output the positions of the gameplay elements to the display
    - Output the GUI to the screen
    - Draw the graphics (textures, particle systems…)
- Makes use of data from:
    - Game controller

2.3. Hardware/software mapping (mapping of sub-systems to processes and computers, communication between computers)

Camera mapping

We need to process the feeds of several cameras on one system. To this end we will be using openCV to create silhouette descriptions of the players. We will use the

Bridge pattern to decouple the abstraction from the implementation, separating the camera detection system from the game, defining a clear interface through which the two will communicate. see also https://sourcemaking.com/design_patterns/bridge.

2.4. Persistent data management (file/database, database design)

-

2.5. Concurrency (processes, shared resources, communication between processes, deadlocks prevention)

-

3. Glossary

-