Developers Manual

Structural overview

Basically the structure follows the UML that can be found in the /doc -directory of the git-repo. Currently there are two activities, one for menus and one for real-time rendering. The Pause-menu is not implemented yet.

The user enters the game in “MainMenuActivity”. Currently it is possible to click around a tiny bit and most importantly to start a race. Technically the activity loads different menu-layouts defined in resources. Different buttons triggers different menus to be opened. This text-based “mini-world” are supposed to be extended so that the player is getting the illusion of clicking around in a “world” with places like “garage”, “car-dealer” and similar. Loading/changing/saving of profiles and settings also occurs in this system, but will be implemented in a way so that it feels like they're not part of the “world”. But everything will be contained in the single activity and the basic principle is just to load different XML’s

When user starts a race they implicitly change activity to what is called the “GameActivity” in the UML. In this activity there are basically three parts: A Game model responsible for the gameLogic, a visual mirror of the model and render/graphics specific stuff.

The model

The model has a level that holds a list of entities. An entity may be anything that is to be visualized, like a vehicle or a power-up. Making an entity is just a matter of implementing the interface and instantiating it. Think of it like a cloud of a lot of different stuff where these two examples are given. For any other object it works the same way.

The view

What happens in the view is basically just a mirroring of what happens in the model. The renderer keeps track of all view-objects that should be drawn. The collection of “EntityView”-objects should correspond to the collection of “Entity”-objects in the model. The View keeps track of the OpenGL primitive that it uses to draw itself, any texture it needs and the mapping of the textures to the primitive.

The render/graphics-stuff

Basically just some stuff to make OpenGL work. The first thing included is a renderer that defines a coordinate system and how it is mapped to the screen and manages the drawing of the visual entities. The other things are graphical primitives (square, triangle, circle) that basically knows their vertices and how to draw themselves.

Digging down the model

The model is the most interesting code to dig down into. The model is controlled by a game-loop that forces model to update iteratively (the game-loop also controls rendering pace). The update request basically flows down the graph of entities held by the model, which update their own state based on last state and any modifications done to that through interaction with user or other objects.

Stuff that move around shall always extend “MovableEntity”, which is an abstract class implementing “Entity” and provides some more functionality for movement. Movement is considered as a manipulation of a 2-dimensional vector (represented by custom class with name “Vector2D”). A good example is the entity Vehicle. Between updates it collects user input and when update-request arrives it manipulates it’s vector, which then can be used by model to check for collisions and move. This is only partially implemented. Collision-checking does not work for instance, but the basic infrastructure is there!

Collision-checking is, as stated above, not working yet, but it will when it does work use “hit-boxes” represented by a class “HitBox”. Every entity has a hit-box and it is basically representing the 2-dimensional space that the graphical representation of the entity fills up.

User input is managed by a class called “Joystick”. It basically takes screen coordinates and then translate what user does to “ActionEvents” and sends it to the controller which passes it on to the model, which process the input. The view render the changed state and then the user responses to that and so the circle is complete.

Speaking about user, there are two classes involved in user-management. They do have confusingly similar names. “Player” and “Players”. Player represent a player. It keeps track of player’s money, of player’s car and so on. Players keeps track of which player is currently set to active. The active player is the one getting progress while user plays.