SpaceRacer

MPD

Ruben Bergshoeff

About the project

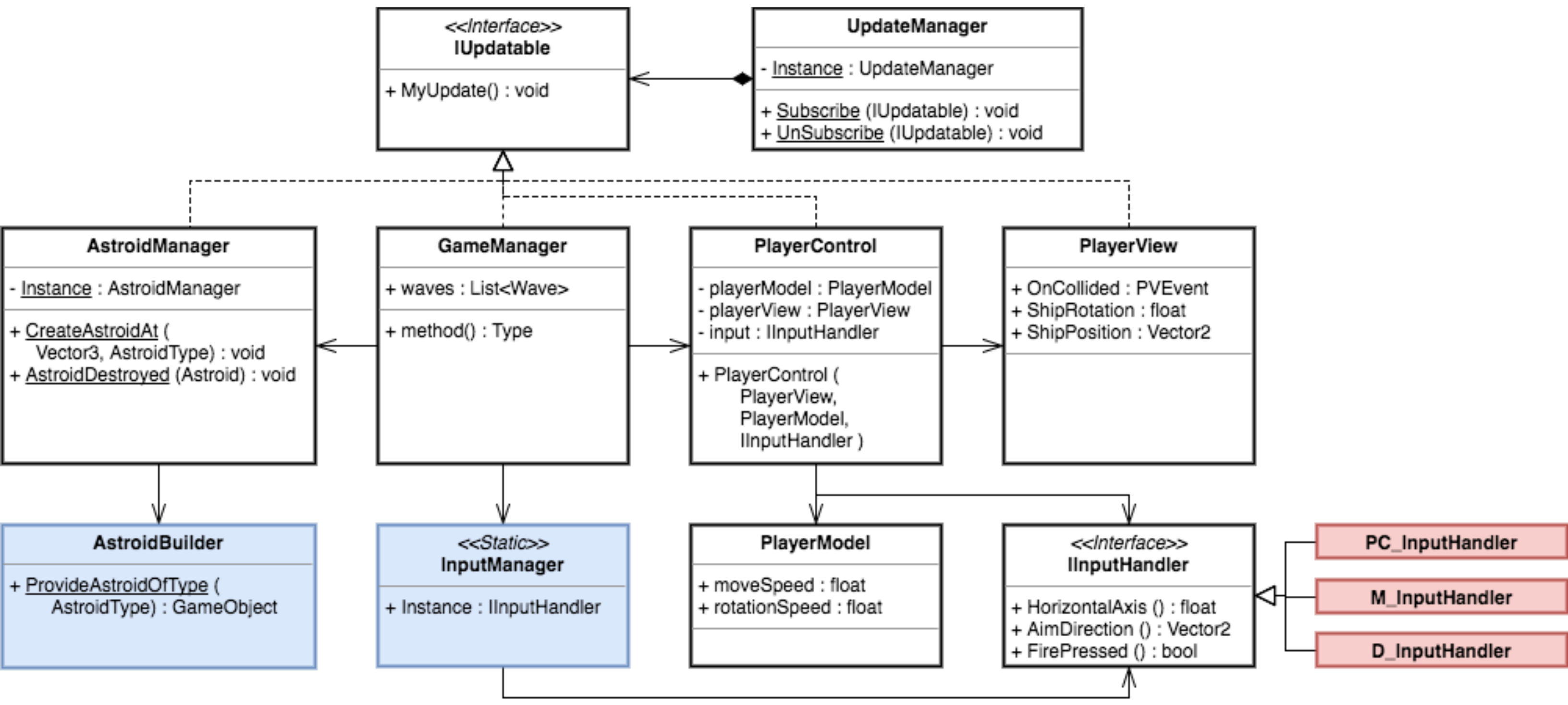
Game Design

Space Racer is a simple endless runner kind of game. You control a spaceship that can fly within a clamped area, evading objects coming your way in waves. In future levels an enemy type could be added that would require fighting.

PC & Android

As for the multi-platform challenge, I decided to develop for PC and Android as the input and graphical capabilities are vastly different.

UML



<<Interface>>
IUpdatable
+ MyUpdate() : void
UpdateManager
- Instance : UpdateManager
+ Subscribe (IUpdatable) : void
+ UnSubscribe (IUpdatable) : void
AstroidManager
- Instance : AstroidManager
+ CreateAstroidAt (Vector3, AstroidType) : void
+ AstroidDestroyed (Astroid) : void
GameManager
+ waves : List<Wave>
+ method() : Type
PlayerControl
- playerModel : PlayerModel
- playerView : PlayerView
- input : IInputHandler
+ PlayerControl (PlayerView, PlayerModel, IInputHandler)
PlayerView
+ OnCollided : PVEvent
+ ShipRotation : float
+ ShipPosition : Vector2
AstroidBuilder
+ ProvideAstroidOfType (AstroidType) : GameObject
<<Static>>
InputManager
+ Instance : IInputHandler
PlayerModel
+ moveSpeed : float
+ rotationSpeed : float
<<Interface>>
IInputHandler
+ HorizontalAxis () : float
+ AimDirection () : Vector2
+ FirePressed () : bool
PC_InputHandler
M_InputHandler
D_InputHandler

Explanation UML : Colors

Blue

Classes highlighted in blue have pre-processor directives to change the functionality based on the platform we're building for.

Red

Classes highlighted in red are scripts that will only work on their designated platform and should not at all be available in builds not of their designated platform. Blue colored classes should take care of this.

Explanation UML : Platform Specific Classes

InputManager

This is a very simple class which, with the use of pre-processor directives, provides the GameManager class with the right input manager.

AstroidBuilder

This class takes care of the building of astroid objects. This is again done with pre-processor directives. For example, when a big astroid needs to be spawned, the PC version will have a different, more detailed mesh, while the Android version will simply spawn a small astroid with a changed scale.

IInputHandler derived classes

These classes handle platform specific input and are instantiated by the InputManager.

Further Relevant Classes

BuildMenuItems

This class is not included in the UML diagram because it is not part of the game code, but it is definitely relevant for development and building, as it switches out relevant files to the Resources folder when we're changing the build target platform.

Designing for both platforms

MVC Setup for Player

To make multi platform development easier, providing a clear view of what was platform specific and what wasn’t, I decided to split up the player object to different parts according to the Model View Controller design pattern. This way I could be sure that platform specific behavior would be in the View part, not in the controller or model. On top of this, splitting up these parts made injection easier: the view is instantiated to a GameObject and is therefore a Monobehaviour, making injection impossible, but PlayerControl is a non-Monobehaviour class. This allows for injection with the correct view and input handler with a constructor.

IUpdatable Interface

As I wanted to have more control over my classes and not have them derive from MonoBehaviour, I had to handle my own updates. I created the IUpdatable Interface to do just that. One singleton class, the UpdateManager, is one of few Monobehaviours, and the only Monobehaviour actually using the Update method. This class updates all other classes, allowing for non-Monobehaviour classes to be updated at the same rate and allowing for more control when which instance is updated.