Thank you for purchasing **2D INFINITE RUNNER TOOLKIT**.

If you have any problems, suggestions, feature requests or questions just let us know and we will try to answer them as soon as possible!

Get in touch via our forum thread or email: mako752@gmail.com

CONTENTS

1. Introduction

Thank you for purchasing 2D Infinite Runner Toolkit! If you have any feedback or questions just let us know and we will try to answer them as soon as possible!
E-Mail: mako752@gmail.com

2D Infinite Runner Toolkit is a feature complete package giving you the chance to create endless runner games of any kind. We invested many hours into this kit to make it as logical and easy to setup and use as possible. It works with Unity3D built-in features and does not need any 3rd Party Tools to get it up and running – of course you can easily expand the kit with any Unity3D supported toolset that you might need.

The main music in the kit is taken from the following site, under Creative Commons 3.0 license. http://opengameart.org/content/jump-and-run-8-bit

2. Version History

1.01:

- Bug fixes

1.02:

- Added support for Unity 3.5
- Replaced unity plane with 2 vertex plane
- Improved layer setup o Improved inspector for GUI Manager
- Improved scene settings
- Managers are now singleton
- Added resolution manager
- New documentation
- Code improvements
- Bug fixes

1.03:

- Fixed several bugs on mobile devices

1.04:

- Fixed: Best distance not showing in main menu
- Fixed: The obstacles are sometimes missing after a crash

1.05:

- Added support for Unity 4.3

2.0:

- The kit structure and the scripts have been recreated from the ground.
- New Input System
- New GUI System
- New Level Manager
- New Level Generator
- New Player Manager
- New Mission System
- New Save Manager
- New Resolution Manager
- New Power-up System
- New Documentation

2.1:

- Audio Manager has been added, with placeholder clips
- Second skin added
- Bug fixes

2.1.1:

- Smaller bug fixes

2.2:

- Added second submarine skin.

2.2.1:

- Improved Resolution Manager

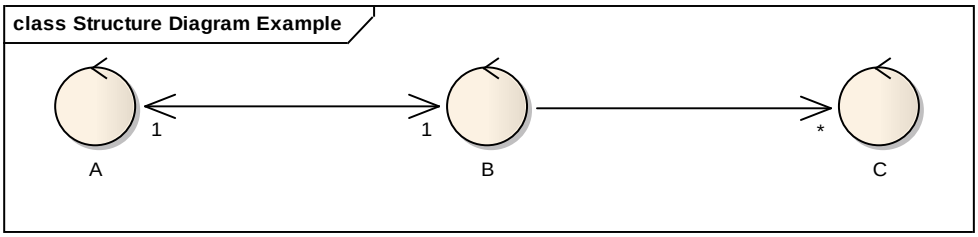
2.3:

- The UI is recreated using Unity's new UI system
- Menu movement and animation is now handled with Unity's animation system

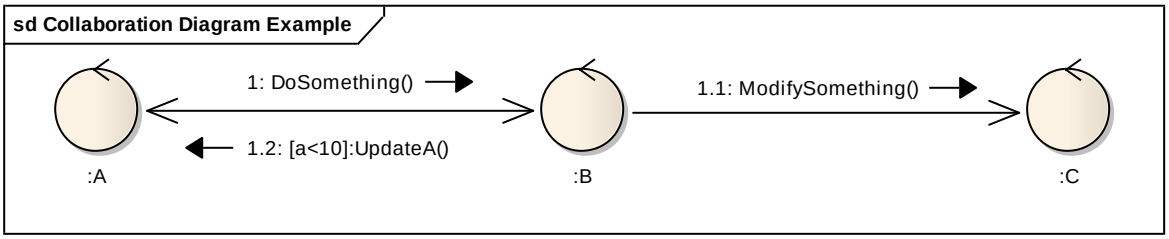
3. UML Overview:

UML (Unified Modeling Language) is a general-purpose modeling language, which is designed to provide a standard way to visualize the design of a system. It has many diagrams, but in this documentation, we will only use 2 (structure and collaboration).

Structure diagrams are used to illustrate the connections between classes. The direction of the communication and the number of classes in it can be specified as well.



In the above example, we can see 3 classes (A, B and C). We can see, that class A can communicate with class B, and class B can communicate with class A and C. Class A and B only have 1 instance (you can see the number 1 in the arrows), but class C can have multiple, unspecified number of instances (you can see the * in the arrow pointing to class C).



In the above example, we can see the same 3 classes. Class A calls the DoSomething() method of class B. Then class B calls the ModifySomething() method of class C. Then, if the a variable of class B is smaller than 10, class B calls the UpdateA() method of class A.

If you want to dig deeper into UML, you can find several guides and examples in the internet.

4. Scene Setup:

Once you have purchased the kit, you should import it to an empty project. Once the importer is finished, open the C# or the JS scene. You can find them under the 2DInfiniteRunnerToolkit/Scenes folder.

The kit works out of the box, so you can press play, and try it out. If you wish to play with the kit on a mobile device, make sure that the Input Manager's Use Touch box in enabled. You can find the Input Manager on the overlord game object.

5. Class Overview

In this section, you can see every class from the kit, and their short description.

Manager Classes:

- GUI Manager: Handles the GUI related functions.
- Input Manager: Scans for player input, processes them, and forwards it to the Player Manager.
- Level Generator: The main level generator class, stores a reference to the level layer classes, and calls their methods when needed.
- Level Manager: The main manager, it calls the other managers' methods when level events occur.
- Mission Manager: Holds the mission classes, forwards mission events to the active missions, when a mission is completed, gives a new one from the mission pool.
- Player Manager: Handles player movement and collision.
- Powerup Manager: Contains the powerup activation functions.
- Resolution Manager: Rescale and reposition the level elements to fit the current resolution
- Save Manager: Save and load level and mission data.

Level Generator Classes:

- Moving Layer: Generates, resets and pauses the moving layers.
- Particle Layer: Generates, resets and pauses the explosion and coin particles.
- Powerup Layer: Generates, resets and pauses the collectible powerups.
- Scrolling Layer: Scrolls and pauses the scrolling layers:
- Torpedo Layer: Generates, resets and pauses the torpedos.

Mission Classes:

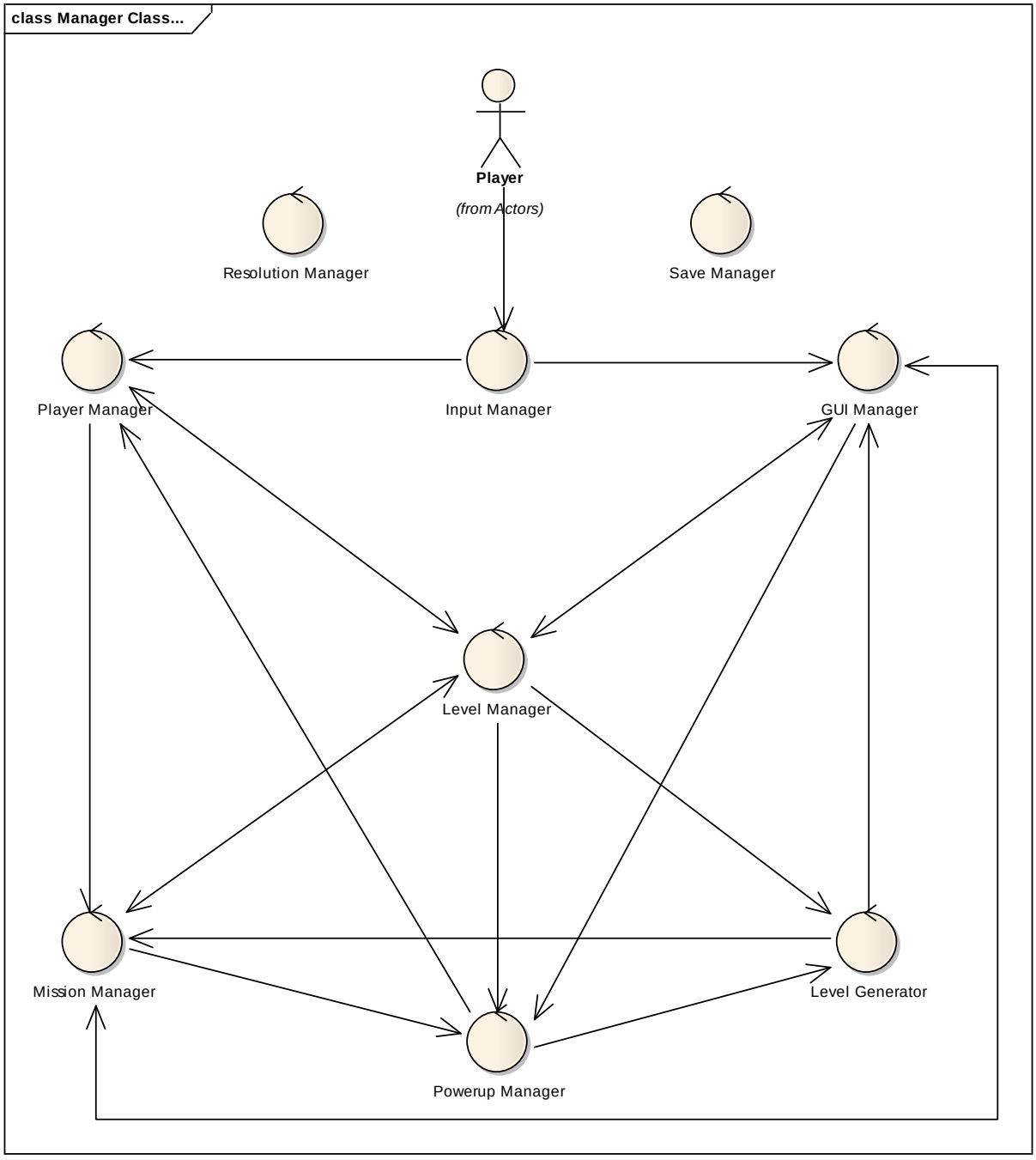
- Mission Template: The main abstract mission template class. The other mission classes derivate from this class.
- Coin Mission: Stores coin mission data, and checks for completion.
- Collision Mission: Stores collision mission data, and checks for completion.
- Crash Mission: Stores crash mission data, and checks for completion.
- Distance Mission: Stores distance mission data, and checks for completion.
- Shop Mission: Stores shop mission data, and checks for completion.

Other Classes:

- Follow: Stays on the target object's position.
- Function Library: A static class library, used by other classes.
- Powerup: Creates the vertical movement of the powerup.
- Set Animation Speed: Sets the animation speed of an animator.

- Set Rendering Layer: Sets the rendering layer and order of an element.
- Sonic Blast: Stores the sonic blast related methods.
- Torpedo Indicator: Creates the vertical movement of the torpedo indicator.

6. Manager Communication Diagram:



You can see 2 classes, which are not connecting to any other classes. The resolution manager does not communicate with other classes, and the save manager is static, so it can be accessed from everywhere.

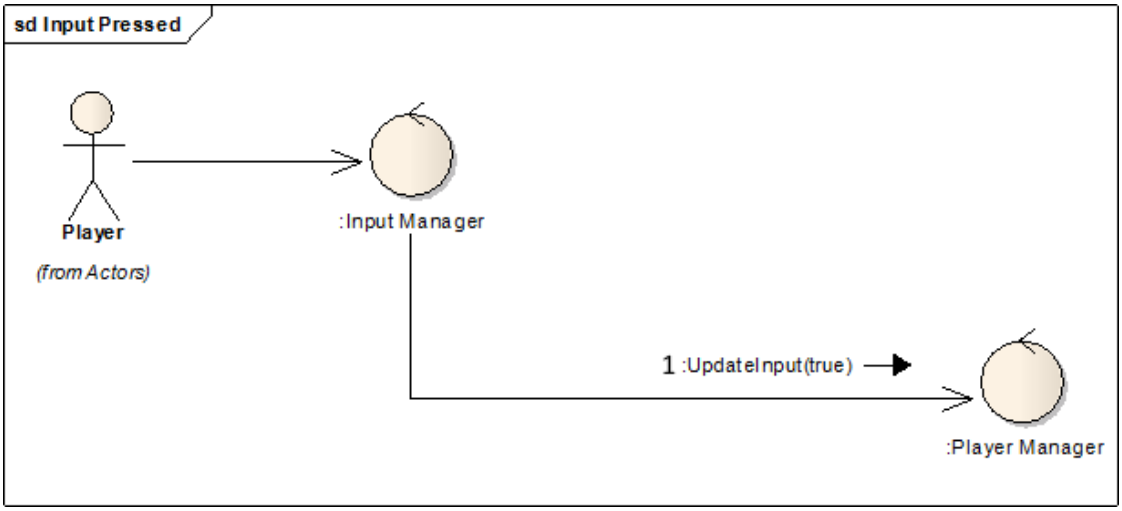
7. Handling User Input:

The user input is detected and processed by the Input Manager. It is assigned to the *PlayTrigger* (child of the canvas), and process the input using the built in event system.

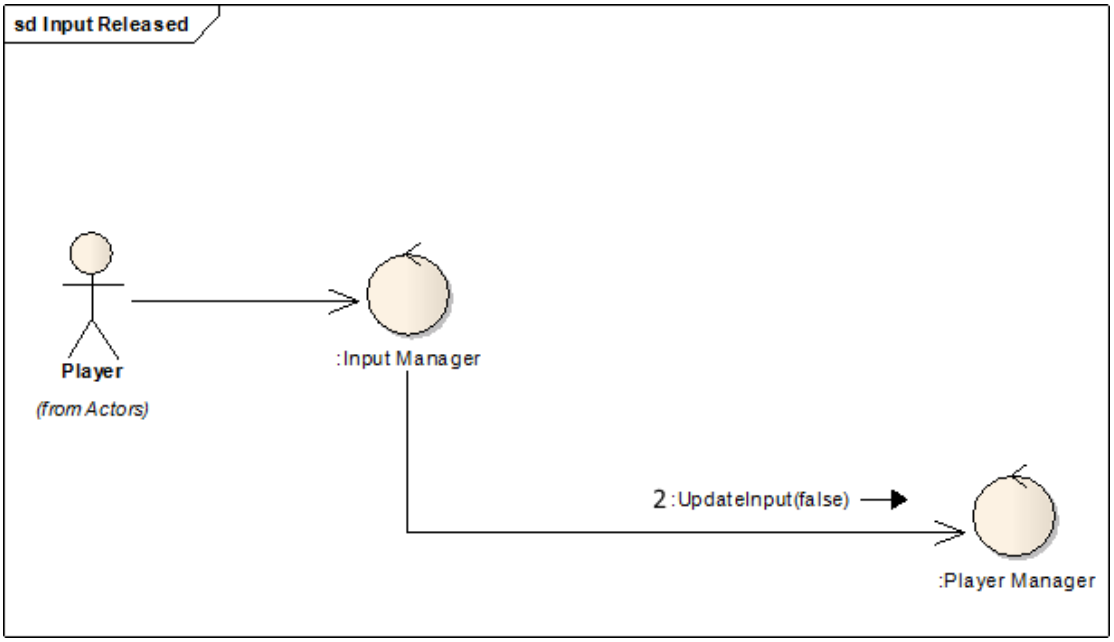
7.1 Switching Between Mouse and Touch Input:

Switching between mouse and touch input is done automatically, using the new event system.

7.2 Input Pressed:



7.3 Input Released:



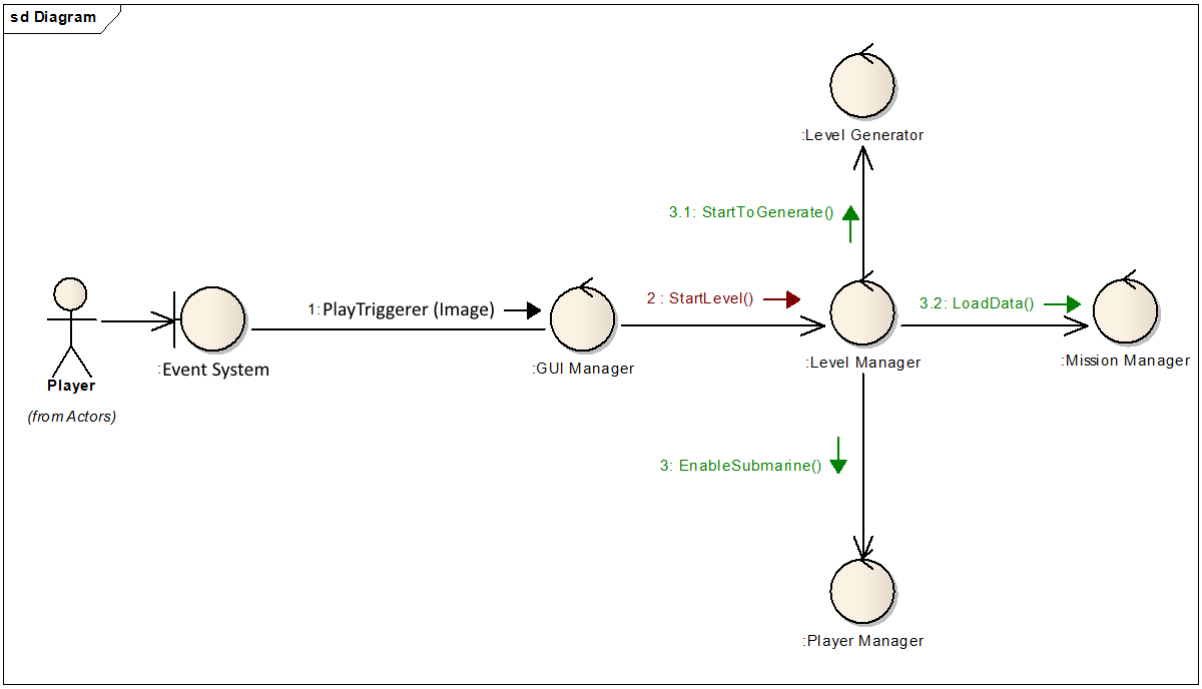
8. GUI Manager:

Handles menu movement, UI logic, and communicates with other managers, when needed. When a player interacts with a button, a function will be called from this manager by Unity's event system. Each and every button has an On Click() property, which can be set in the inspector. This defines what to do, when the player has clicked on the button.

9. Level Events:

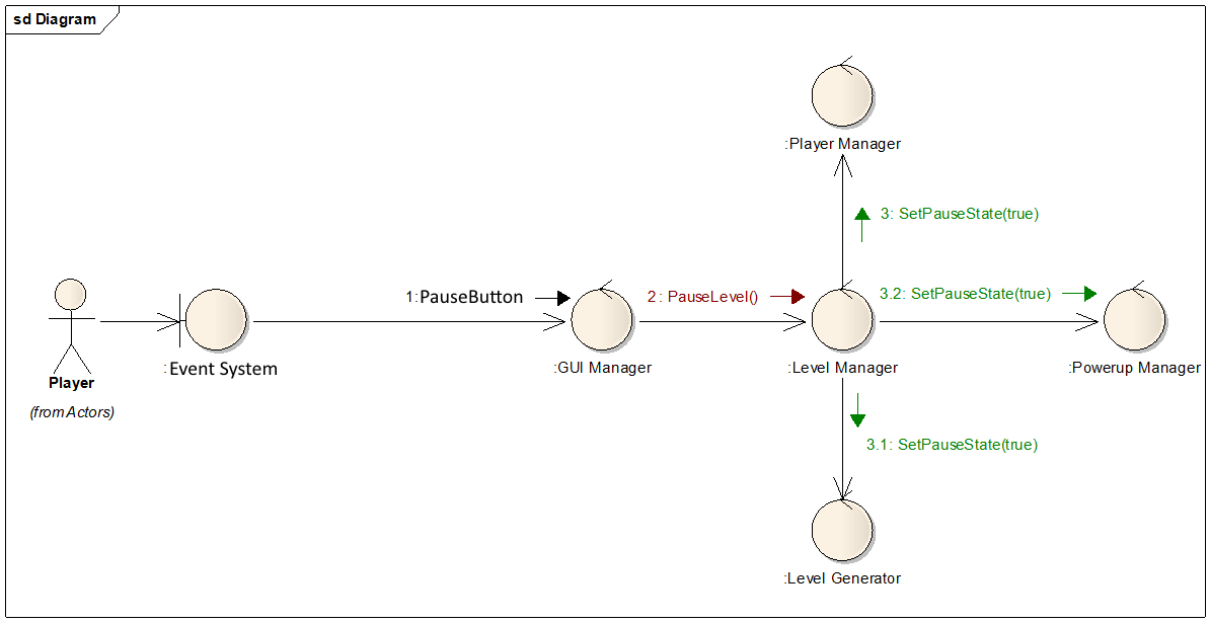
You can find the collaboration diagrams and descriptions of the main level events.

9.1 Starting the Level:



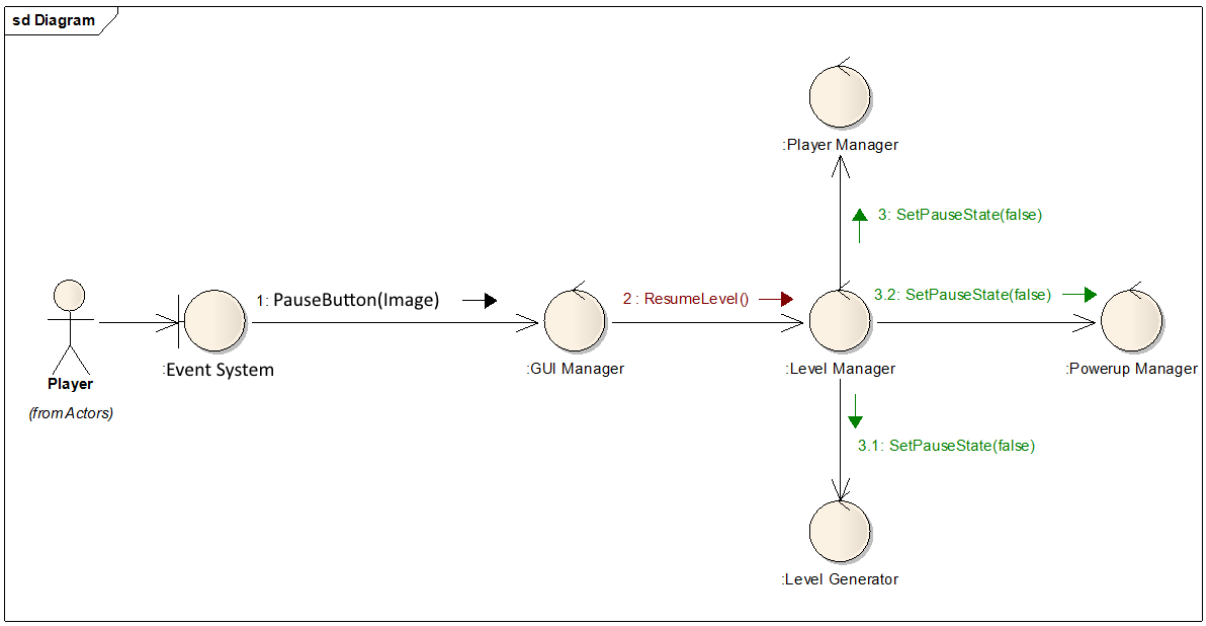
If the player clicks on the level, while in the main menu, the level will start. First, the GUI Manager is notified, which disables the main menu, and activates the main GUI, then notifies the Level Manager. Then the Player Manager enables the submarine, the Level Generator starts the generation process, and the Mission Manager loads the mission data.

9.2 Pausing the Level:



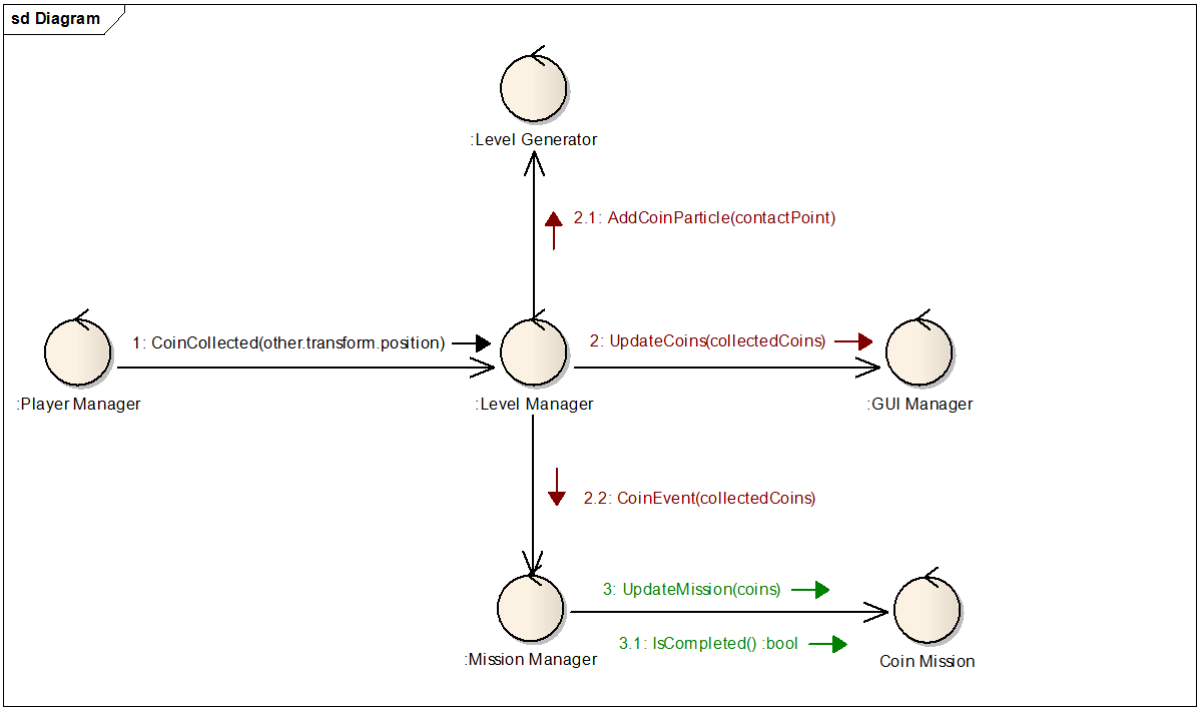
If the player presses the pause button, then the main GUI is disabled, then pause menu is showed. Meanwhile, the Level Manager is notified, and it pauses the Player Manager, the Level Generator, and the Powerup Manager.

9.3 Unpausing the Level:



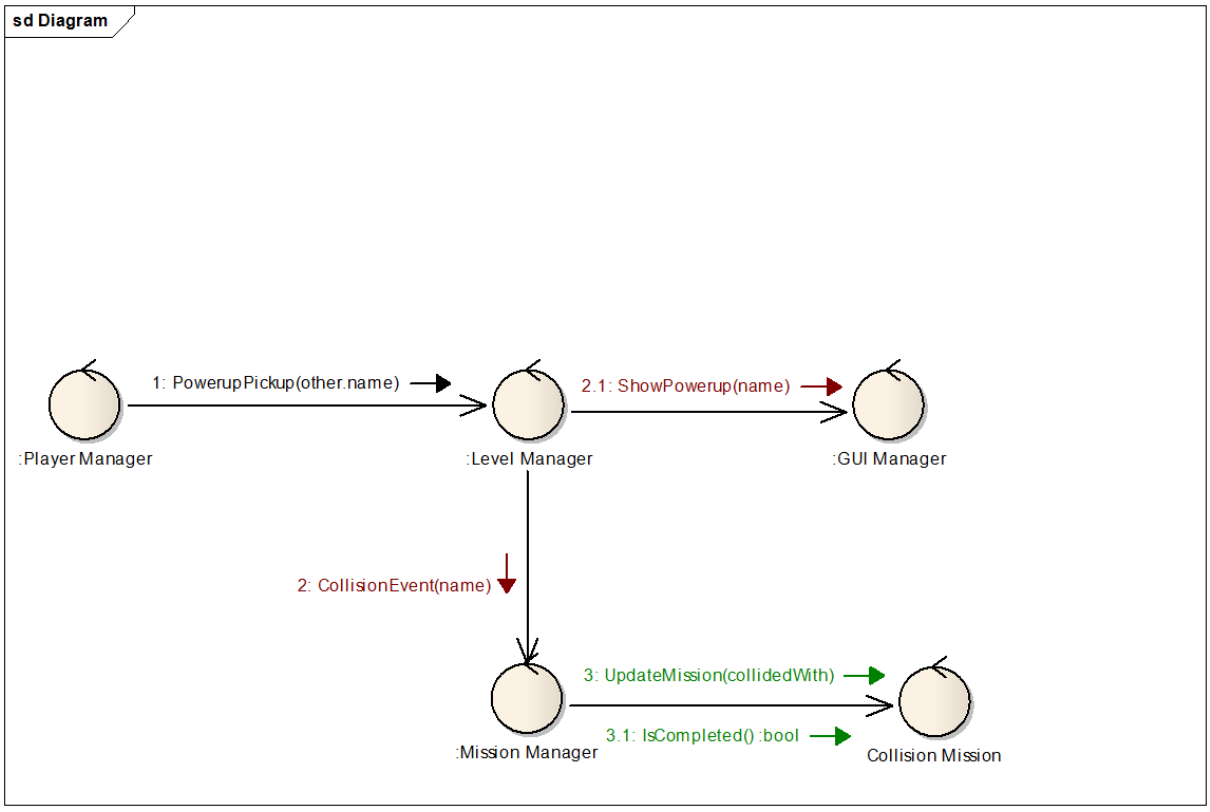
If the player presses the resume button, then the pause menu is hidden, and the main GUI is activated. Meanwhile, the Level Manager is notified, and it unpauses the Player Manager, the Level Generator, and the Powerup Manager.

9.4 Collecting a Coin:



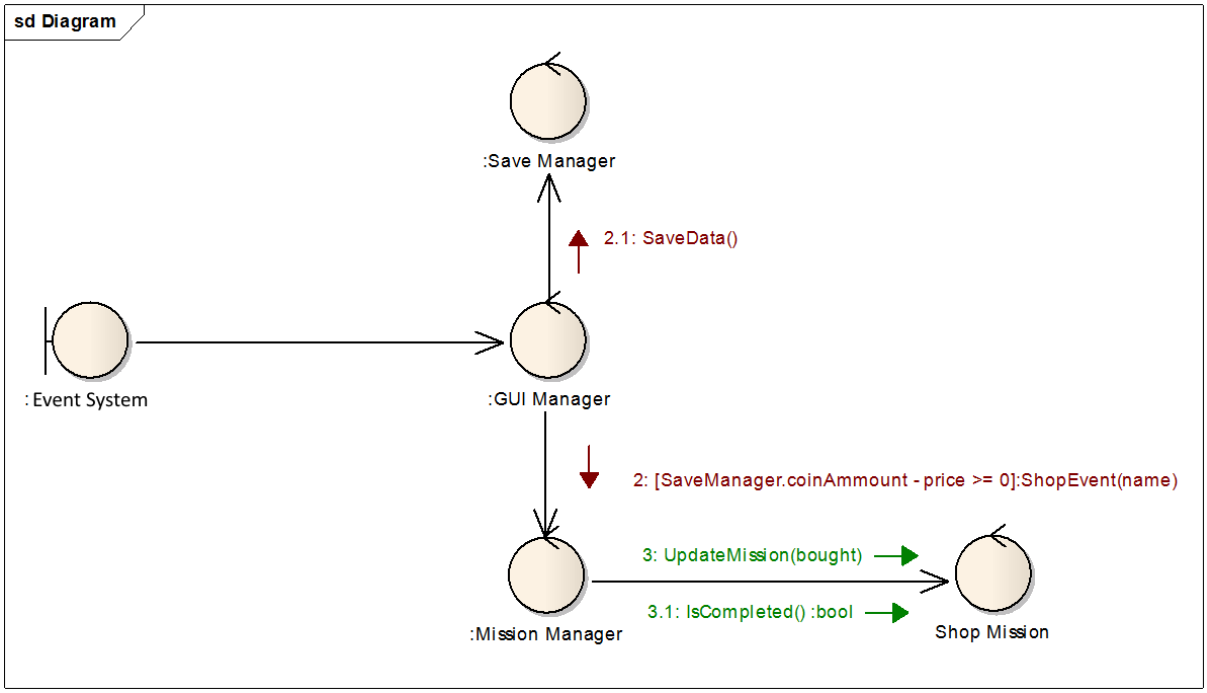
When the player collides with a coin, its collider and renderer is disabled, and the Level Manager is notified. Then it adds a coin particle to the coin's position, updates the main GUI, and notifies the Mission Manager.

9.5 Collecting a Powerup:



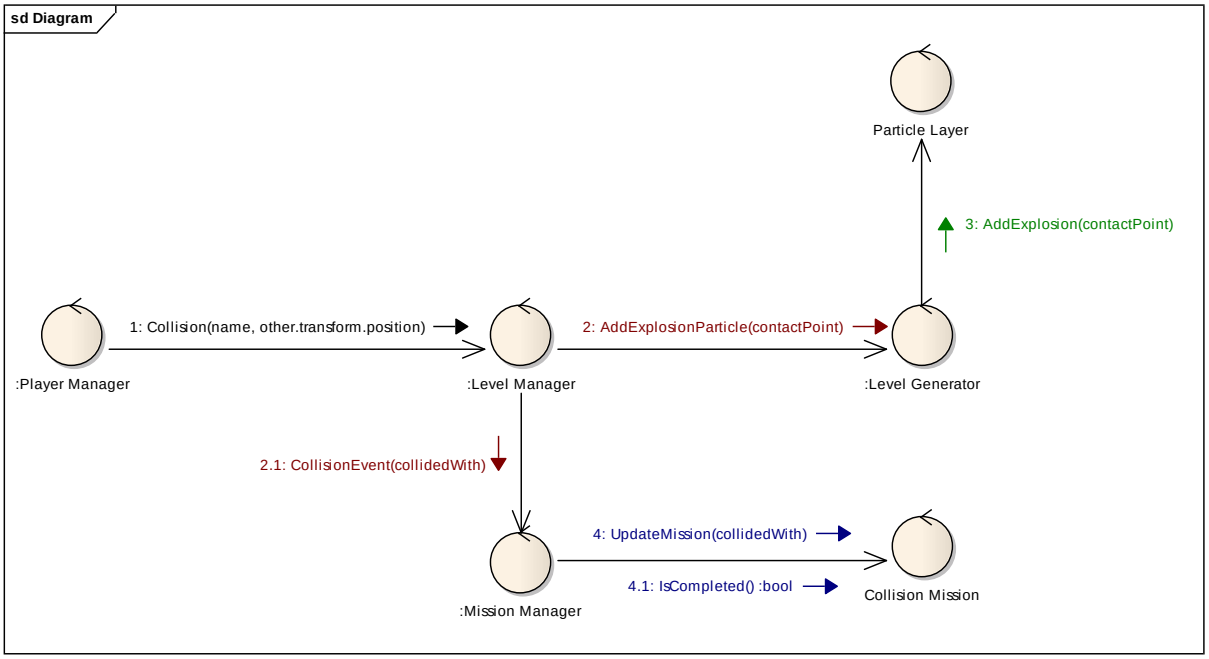
When the player collects a powerup, its collider and renderer is disabled, and the Level Manager is notified. Then the powerup's icon is shown in the main GUI, and the Mission Manager is notified as well.

9.6 Purchasing a Powerup:



Note: The "BuyPowerup" is not the real button name. In the kit, it is called "BuySpeed" or "BuyShield" or "BuySonicBlast" or "BuyRevive", depending on the powerup.

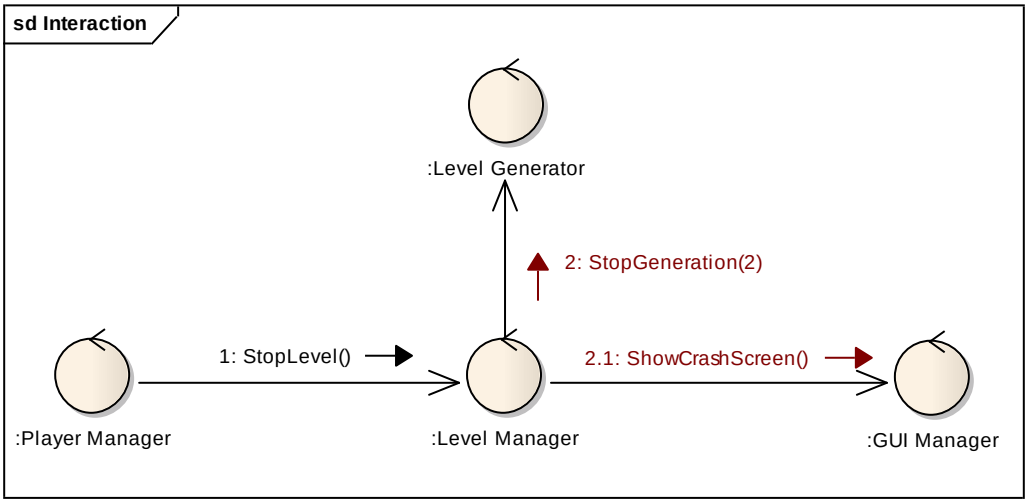
9.7 Colliding with Obstacle:



When the player collides with an obstacle, its collider and renderer is disabled, and the Level Manager is notified. Then it adds an explosion particle at the obstacle's position, and notifies the Mission Manager.

If the player had an active shield, it is collapsed. If the player is vulnerable, then it starts to sink. Once it reaches the sand, the below event will happen.

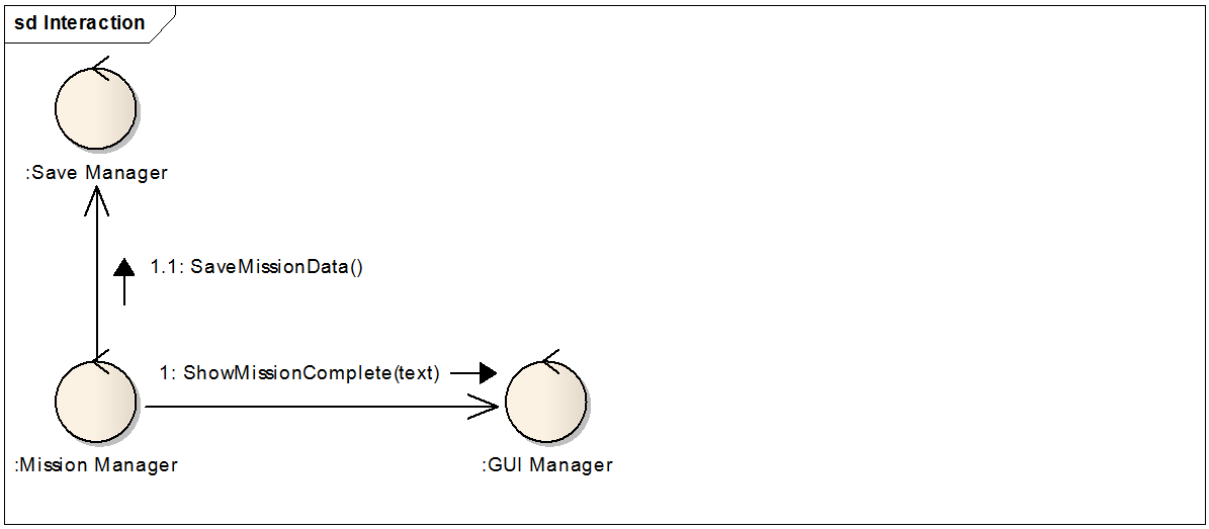
9.8 Player Crashed:



Once the player has crashed to the sand, the Level Manager is notified. Then it stops the level generator's speed to zero in 2 seconds. After that, the GUI Manager shows the crash screen.

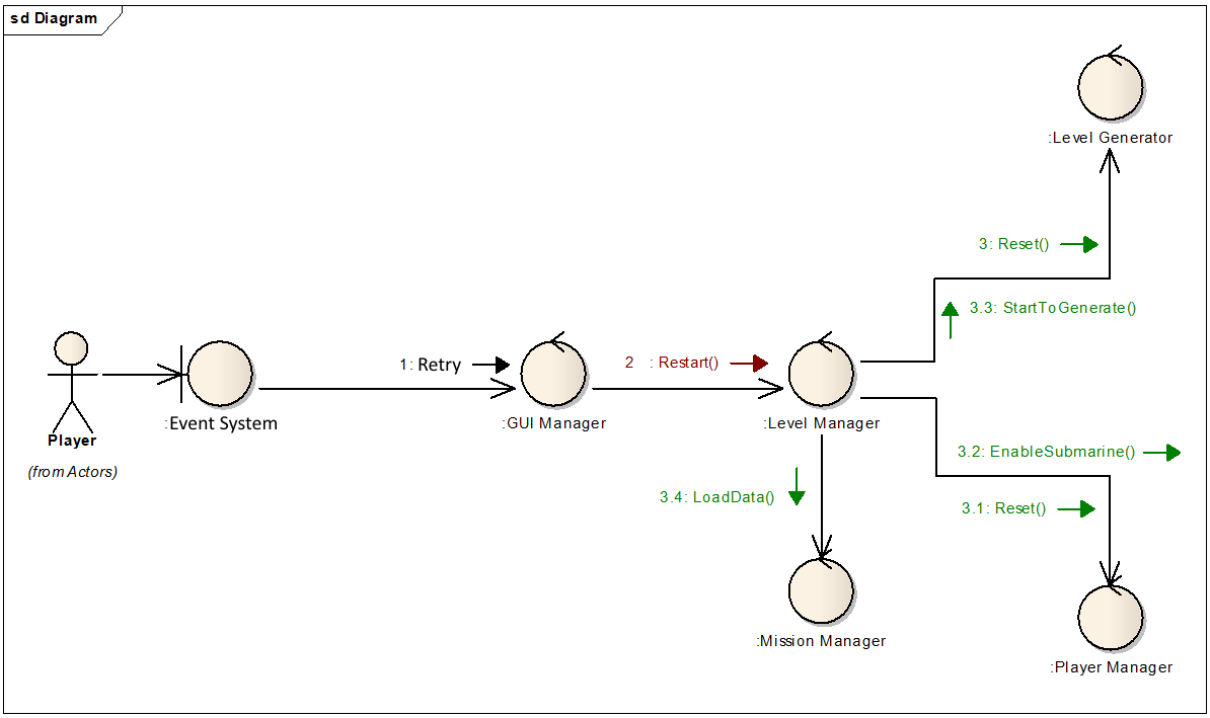
If the player has an active revive, then the revive will be shown. If not, then the finish screen will be shown.

9.9 Mission Completed:



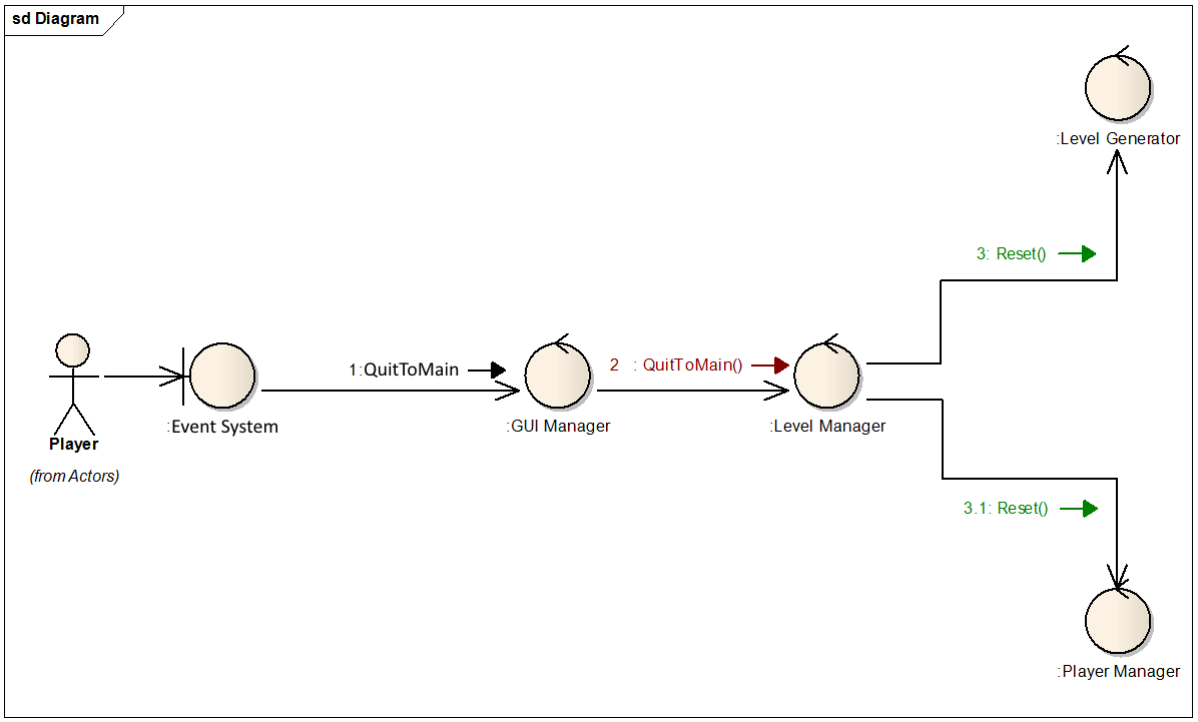
If a mission is completed, the mission data string is updated and saved, while the mission complete notification is shown.

9.10 Restarting the Level:



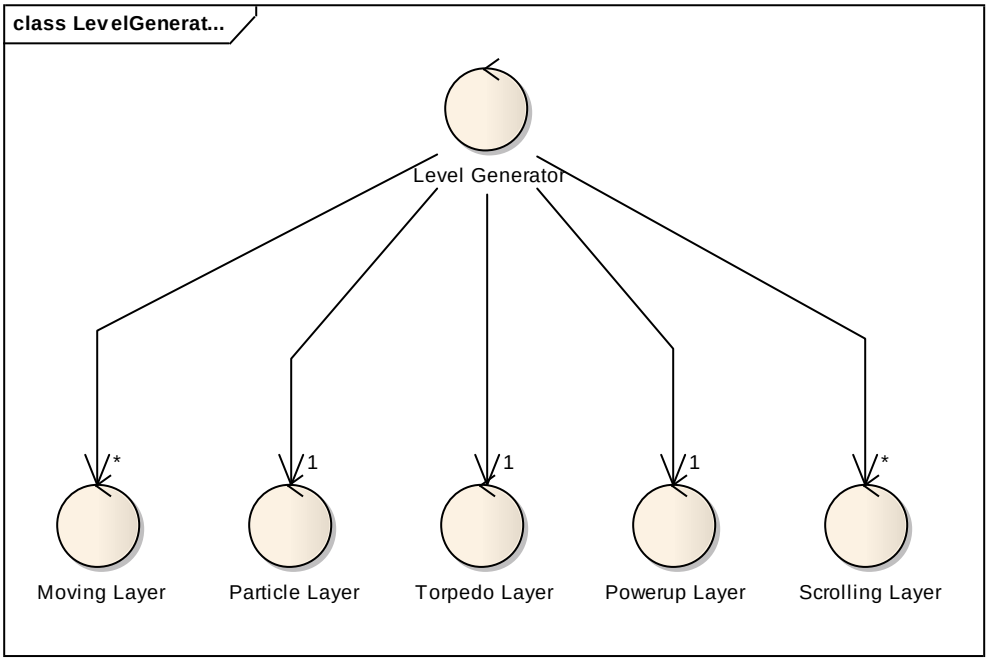
If the player presses the restart button, the finish or the pause menu is hidden, depending on where the player pressed the button. Then the main GUI is reset, and the Level Manager is notified. Then the Level Generator and Player Manager is reset, then restarted, as well as the mission data is reloaded.

9.11 Quit to Main Menu:



If the player presses the quit button, the finish or the pause menu is hidden, depending on where did the player press the button. Then the Main Menu is activated, and the Level Manager is notified. Then the Level Generator and Player Manager is reset.

10. Level Generator:



The Level Generator holds a reference to the layer scripts. The manager's primary role is to forward the incoming notifications to the layer classes, and to manage the scrolling speed.

10.1 Scrolling Speed:

The speed of each layer is defined by their starting speed (which can be set in each layer), and the speed multiplier. This multiplier is 1 at the beginning of the level, and increasing by 0.005 at every frame. Then this multiplier is forwarded to the level layers, and their speed is calculated by the following: startingSpeed x speedMultiplier.

This enables the kit to increase the scrolling speed over time, while the layers keep their relative speed to each other.

10.2 Starting the Generator:

Each moving layer has a StartGenerating() method, while the scrolling layers are started by simply unpausing them. When the Level Generator's StartToGenerate() method is called, then it calls these methods, starting the entire system.

10.3 Stopping the Generator:

The generator can be stopped, by setting the speed multiplier to zero.

10.4 Resume the Generator After a Revive:

When the generator is stopped, the last speed multiplier is saved. After a revive, the speed multiplier is moved from zero to the last value under 2 seconds.

10.5 Moving Layer:

The moving layers are the most common layers in the kit, like the background elements and obstacle groups. It spawns new elements in the inspector defined rate multiplied by the speed multiplier, and moves them across the level, until they reach the reset position. Then they are disabled and placed back to the starting position, waiting to be activated again.

10.6 Torpedo Layer:

The Torpedo Layer is a modified Moving Layer. When it generates a torpedo, it places an indicator in a random position for a given time. Once the time is up, the indicator is removed, and the torpedo is launched.

10.7 Particle Layer:

The Particle Layer adds the coin and explosion particles to the level to the given position. Once the particle has been played, it is removed from the level.

10.8 Powerup Layer:

The Powerup Layer is a modified Moving Layer. It works the same as the Moving Layer, but when spawning an element, it activates the particle's trail as well.

10.9 Scrolling Layer:

The Scrolling Layer is different from the rest of the layers, it does not move elements. It scrolls the texture of the element, by setting the material's offset on the X axis.

11. Powerups:

Powerups are special items, which give some kind of boost to the player. They can be bought from the shop or collected during gameplay. The active, ready to use powerup icons can be seen in the bottom left of the screen. If an input is pressed on one of the icons, then the powerup is activated and the icon is disabled.

11.1 Powerup Manager:

The Powerup Manager calls the other manager's powerup related methods. More details below.

11.2 Extra Speed:

Once activated, it sets the Level Generator's speed multiplier value to a given number, and makes the player invulnerable for a given time. Once the time is up, the Level Generator returns to the last speed multiplier, and the player is vulnerable again.

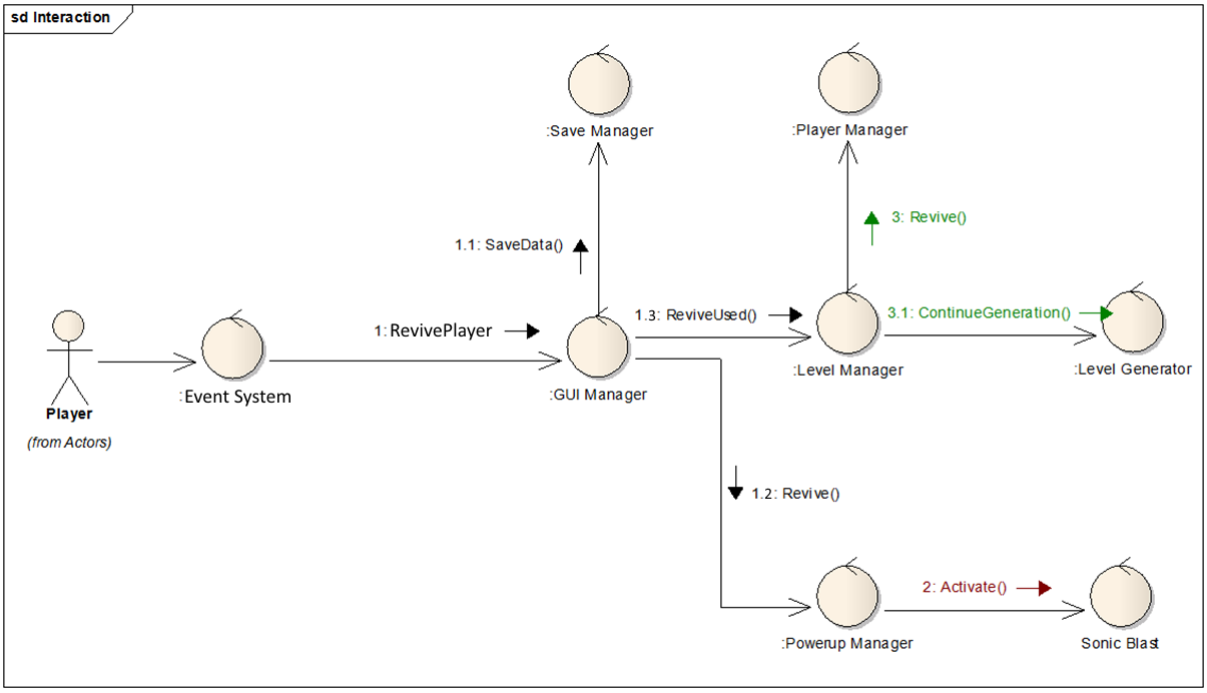
11.2 Shield:

Once activated, the player receives a shield around the submarine, protecting it from the next collision. If the player collides with an obstacle while the shield is active, the shield will collapse.

11.3 Sonic Blast:

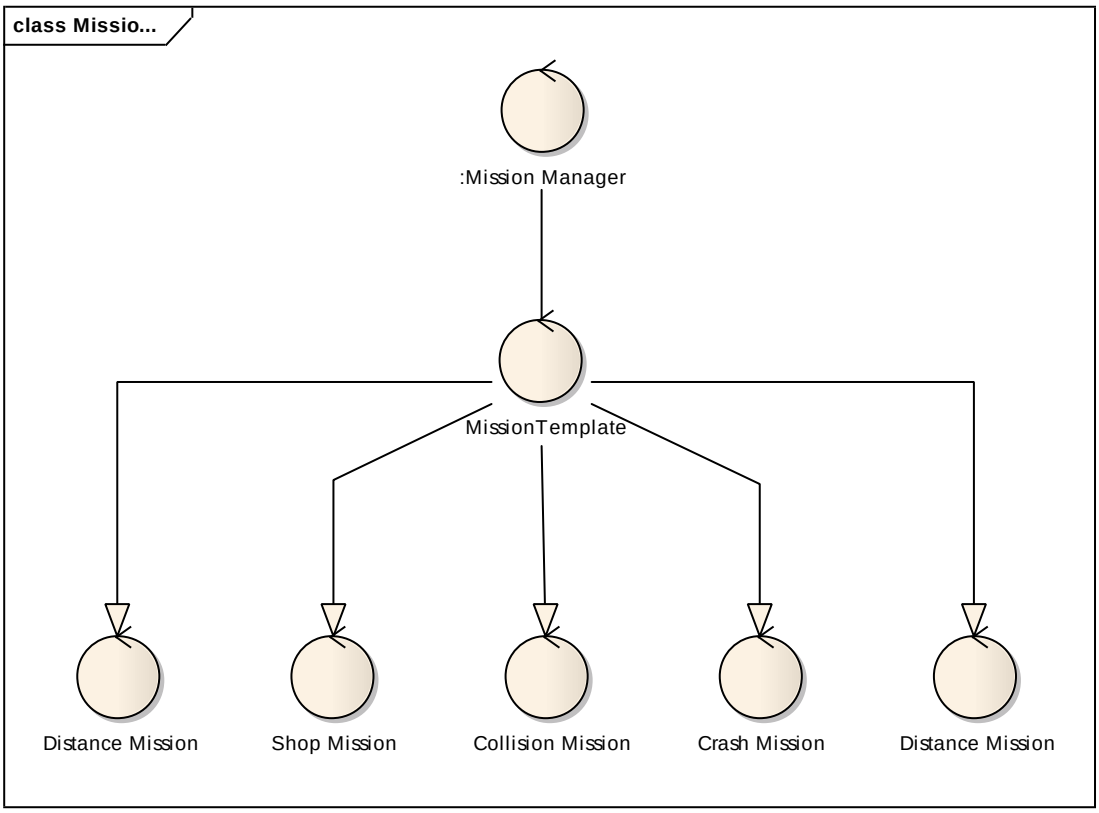
Launches a sonic wave across the screen, from left to right. The wave destroys every obstacle it hits.

11.4 Revive:



If the player has a revive when crashing to the ground, then the player can activate it, and continue the level.

12. Mission Manager:



Each mission type has its own script, and every mission script derivates from the Mission Template abstract script, which contains the method definitions. The Mission Manager stores the actual missions via the Mission Template.

12.1 Storing mission stats:

Mission related data are stored by the Save Manager.

Mission completition data is stored in a string. The length of the string is equal to the number of missions. Each element in the string can have a value of 0 and 1. 0 mean that the mission is not completed, and 1 means that the mission is completed. So, if the data string has a “11101” value, it means that there are a total of 5 missions, where every mission is completed, expect mission 4.

The Save Manager also stores the active mission ID, and the active mission status.

12.2 Mission Types:

- Distance missions: Reach a given distance in one of the following circumstances:
  - In one run: Reach the distance in a single level
  - In multiple run: Travel the given distance in multiple runs
  - No coins: Reach the given distance without collecting coins
  - No powerup: Reach the given distance without using powerups.
- Coins missions: Collect the given amount of coins in one run, or in multiple run.
- Crash Missions: Crash before or after a given distance, or in a given area.
- Shop Missions: Buy the given powerup
- Collision Missions: Collide with the given obstacle or powerup

13. Save Manager:

The Save Manager is a static class, used to save the coin amount, best distance, owned power-ups and mission data is stored by using Unity's built-in PlayerPrefs.

14. Resolution Manager:

The Resolution Manager is used to reposition the hangar, the submarine, and the torpedo indicators for the current aspect ratio.