Assignment 2 Pang!:

Group: -1

Jurriaan den Toonder: 4431324
Erik Wiegel: 4476328
Govert de Gans: 4491955
Jaap-Jan van der Steeg: 4456130
Tim Rietveld: 4472926

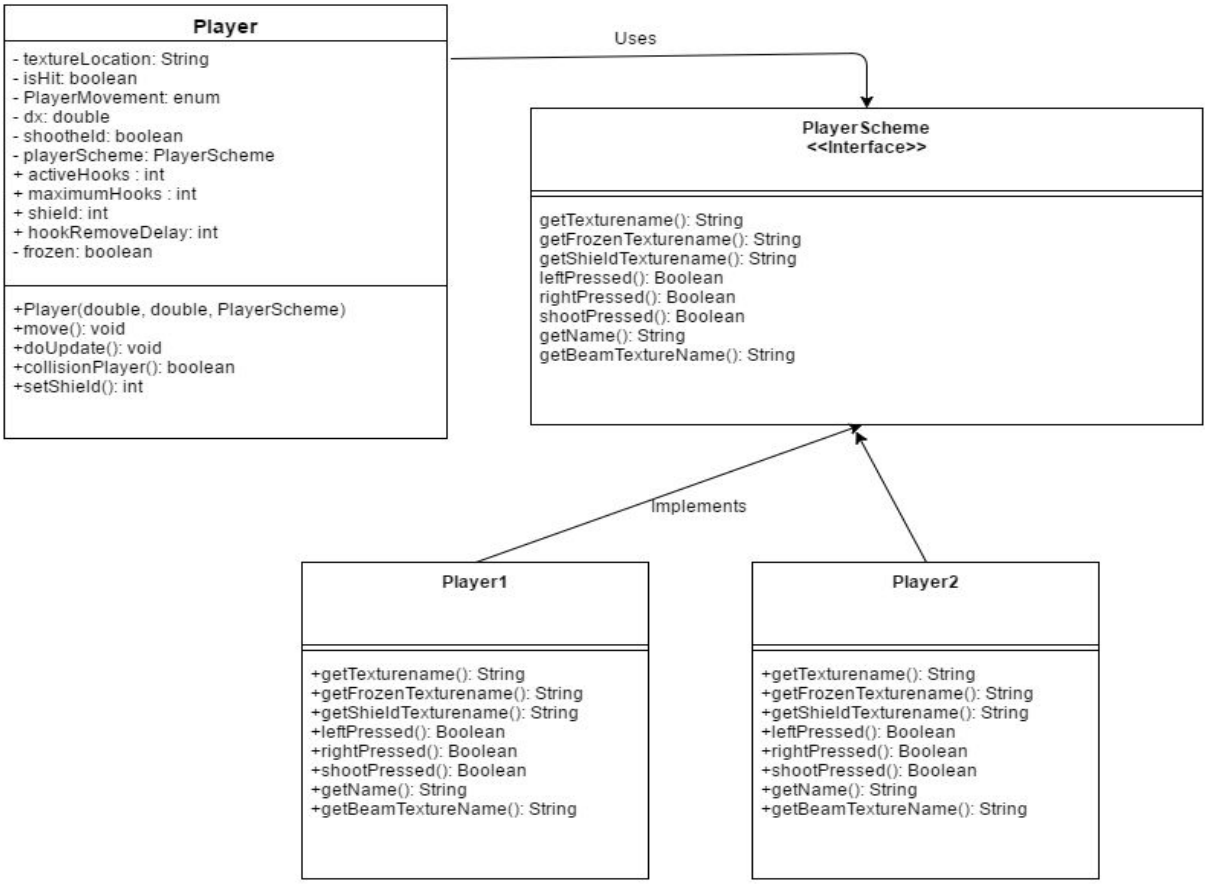
Exercise 1a:

1. Write a natural language description of why and how the pattern is implemented in your code (5 pts).

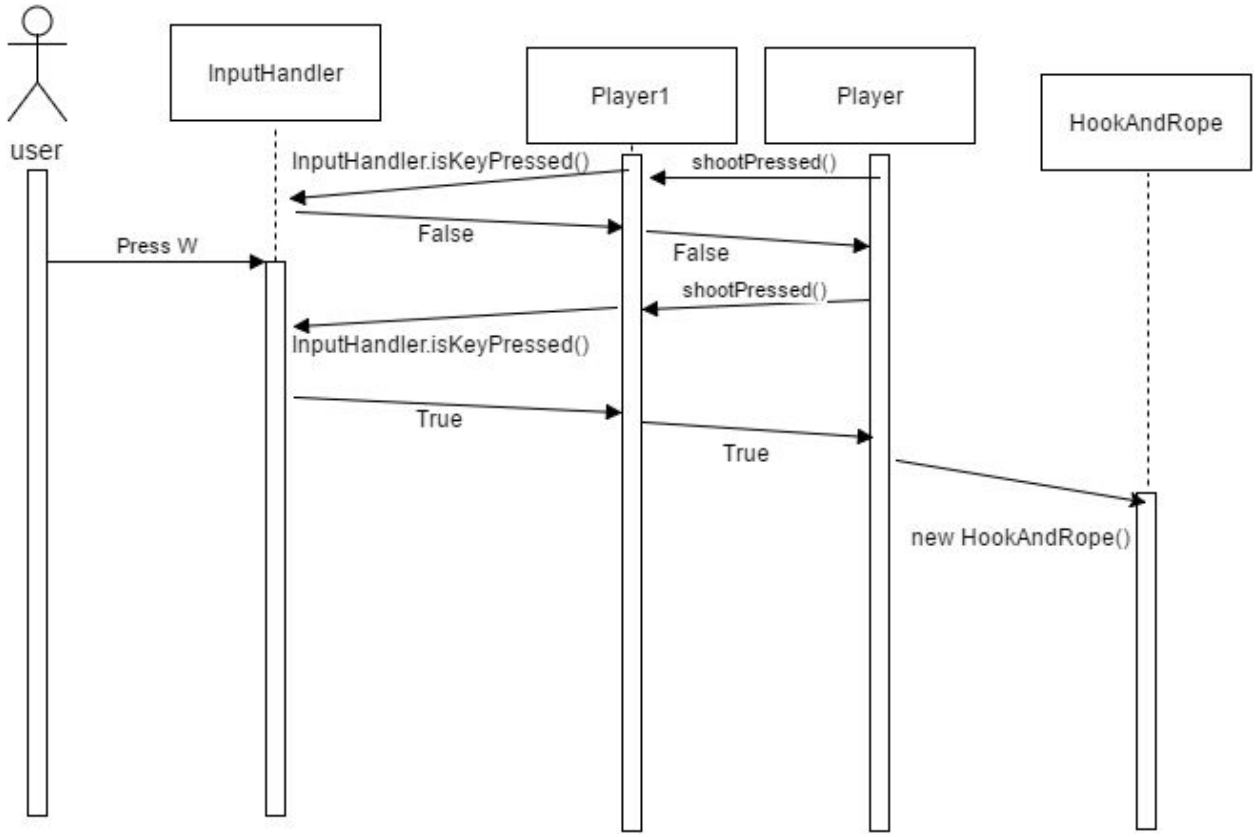
Because we want both players to work with different controls, but do not want to have different player objects, (because then we would have to change 2 classes to change player behaviour,) we used the strategy design pattern to abstract the player controls.

Things related to keypresses are all handled by these PlayerScheme classes, which the Player can simply call upon. We only have to assign the right PlayerScheme to the player when creating it.

2. Make a class diagram of how the pattern is structured statically in your code (5 pts).



3. Make a sequence diagram of how the pattern works dynamically in your code (5 pts).



Exercise 1b:

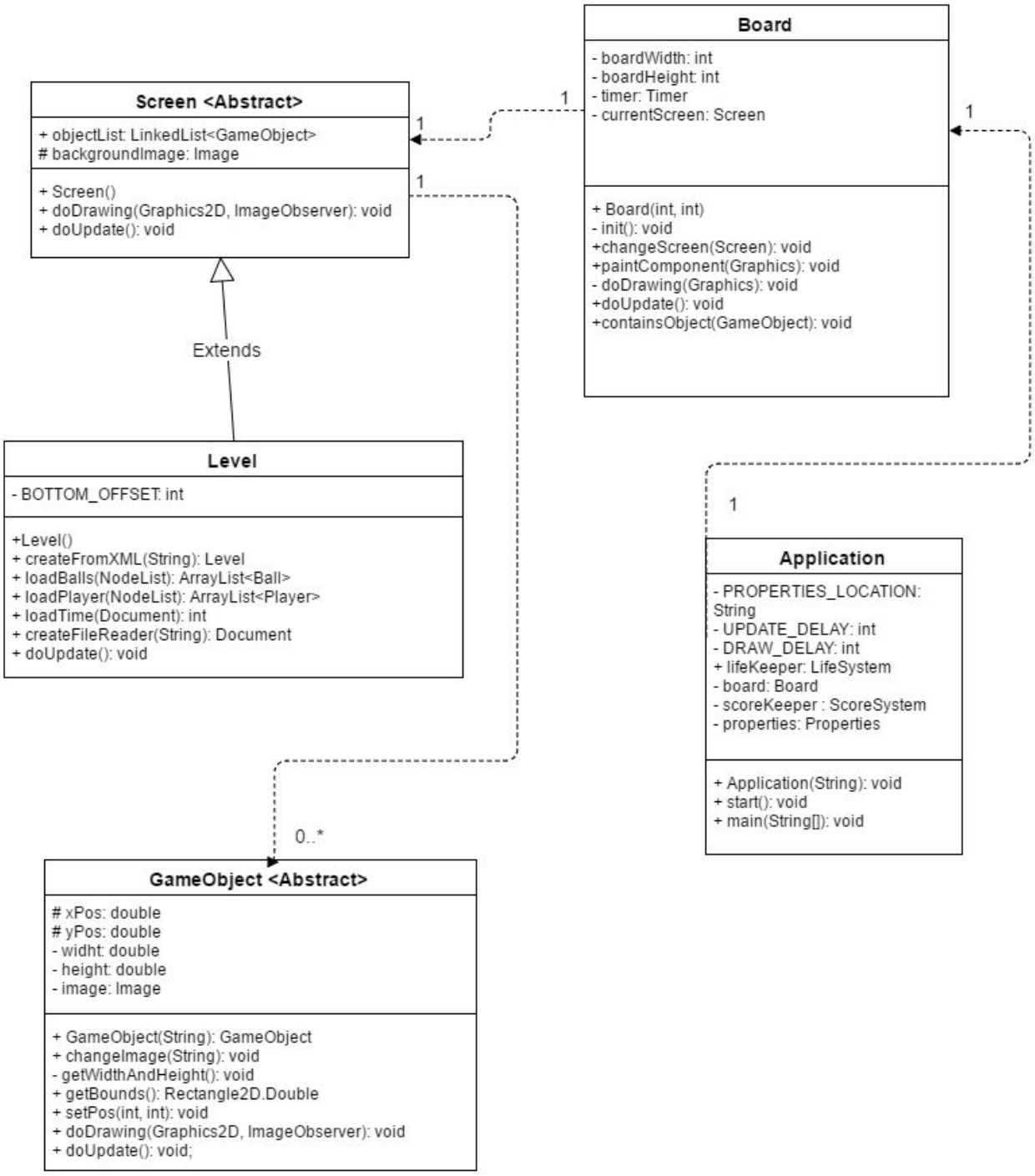
1. Write a natural language description of why and how the pattern is implemented in your code (5 pts).

Because we want to have an easy way to draw and update all GameObjects when an update delay has passed, we use Screen as an subject and GameObjects as observers.

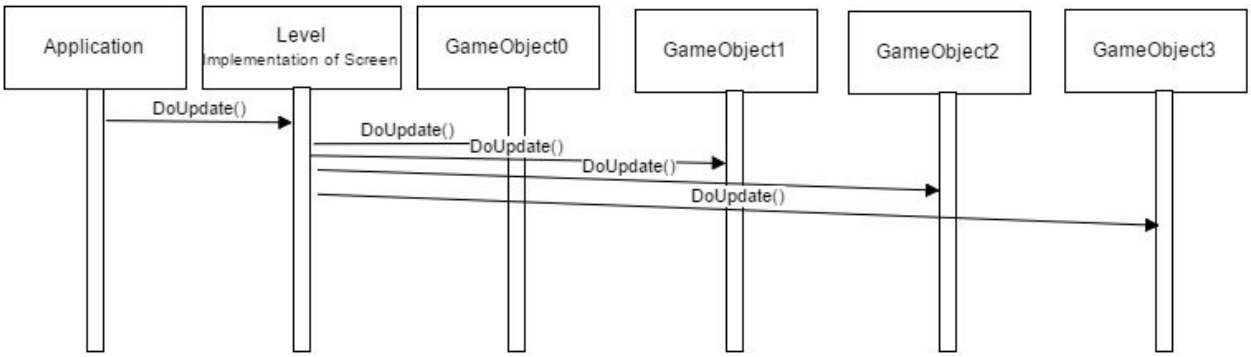
GameObjects all have the doUpdate() method, Screen calls doUpdate on all its registered GameObjects when it's time for an update. Application calls doUpdate on the screen when it's time to update.

Screen is an Abstract class and has implementations that add functionality for more specific situations. (e.g. level also checks if the level has ended besides updating only gameObjects.)

2. Make a class diagram of how the pattern is structured statically in your code (5 pts).



3. Make a sequence diagram of how the pattern works dynamically in your code (5 pts).


Application
Level
Implementation of Screen
GameObject0
GameObject1
GameObject2
GameObject3
DoUpdate()
DoUpdate()
DoUpdate()
DoUpdate()
DoUpdate()

Exercise 2:

Requirements
Pang!: Power-Ups:
Group: -1

Jurriaan den Toonder: 4431324
Erik Wiegel: 4476328
Govert de Gans: 4491955
Jaap-Jan van der Steeg: 4456130
Tim Rietveld: 4472926

Functional Requirements:

Must have:

- Power-ups should work on individual players when applicable.
- We want a Power-up that gives you a customizable amount of lives.
- We want to implement these classic Pang power-ups:
    - Dynamite: Classic Dynamite pops all balls to their smallest size, we want to pop them just once.
    - Double Wire: Allows 2 Hooks to be shot, we want to allow a customizable amount of hooks.
    - Freeze Time: Freezes Time and makes the player invulnerable.
    - Power Wire: This cause the Hook to not immediately disappear when it hits the ceiling. We want the delay to be customizable.
    - Invincibility: This causes the player to not lose a life when being hit by a ball. We want the amount of hits a player can take to be customizable, multiple of this drop should stack, i.e. when we get 2 drops that allow us to take 1 hit, we should be able to take 2 hits.

Should have:

- Power-ups (except the live drop) should only last until the end of the level
- The Drop chance of a Power-up should be easily adjustable.
- We should have an revival power-up for multiplayer, i.e. when one player has died, and the other player gets this power-up, the death player should be revived.

Could have:

- The classical power-up: Slow Time, this would slow the movement speed of the balls.

Won’t have:

- The classical power-up: Vulcan missile, this allows you to shoot an unlimited amount of bullets. (Which would make our game far too easy.)

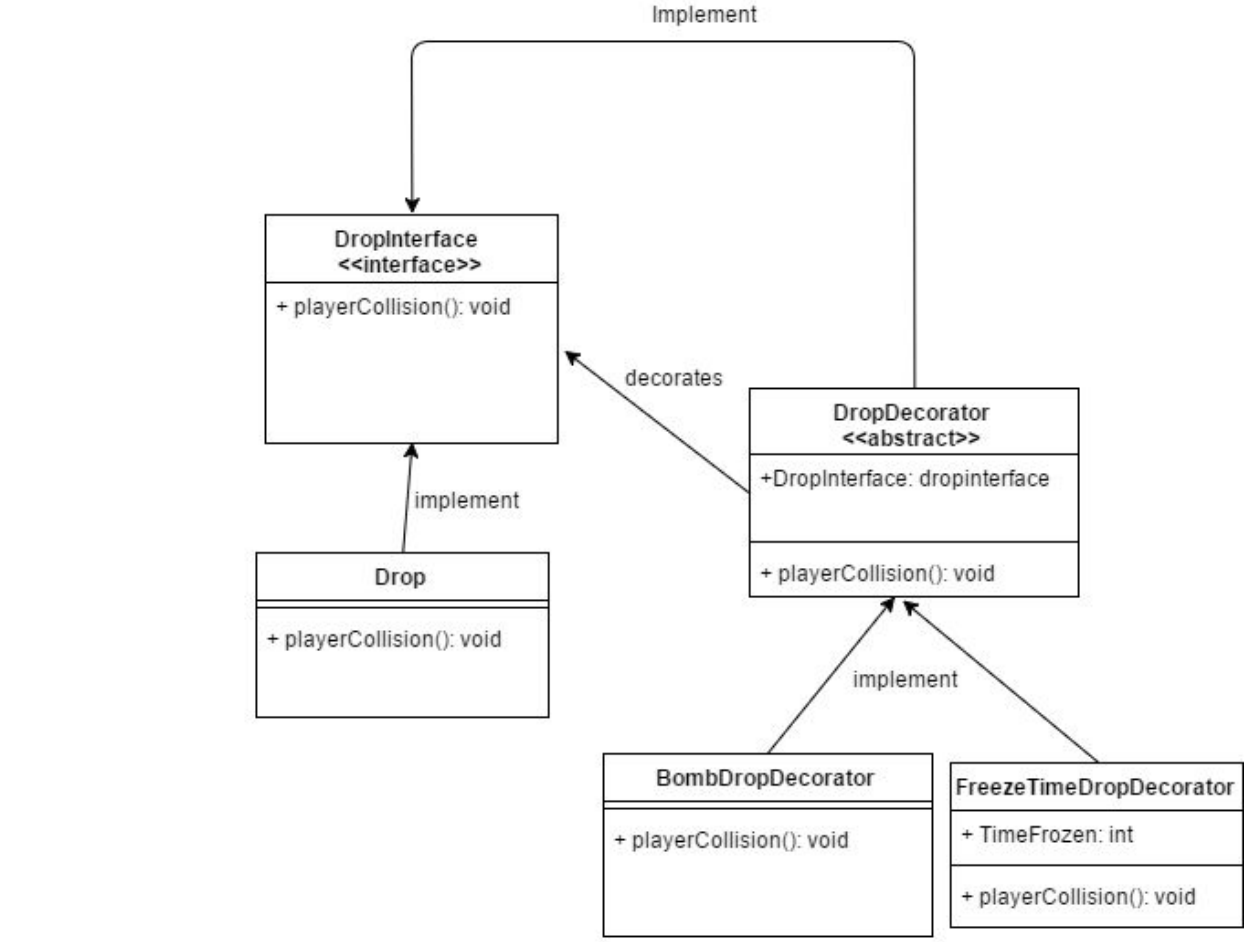
Conceptual objects:

- Drop - And it’s many extensions.
- Player

Non-functional Requirements

- Images used within in the game should be free-to-use.
- Power-ups should not be explained in-game, but it should always be visible in some manner what is happening when a power-up is activated.
- Power-ups should all have a different image, which should somewhat be related to what they do.

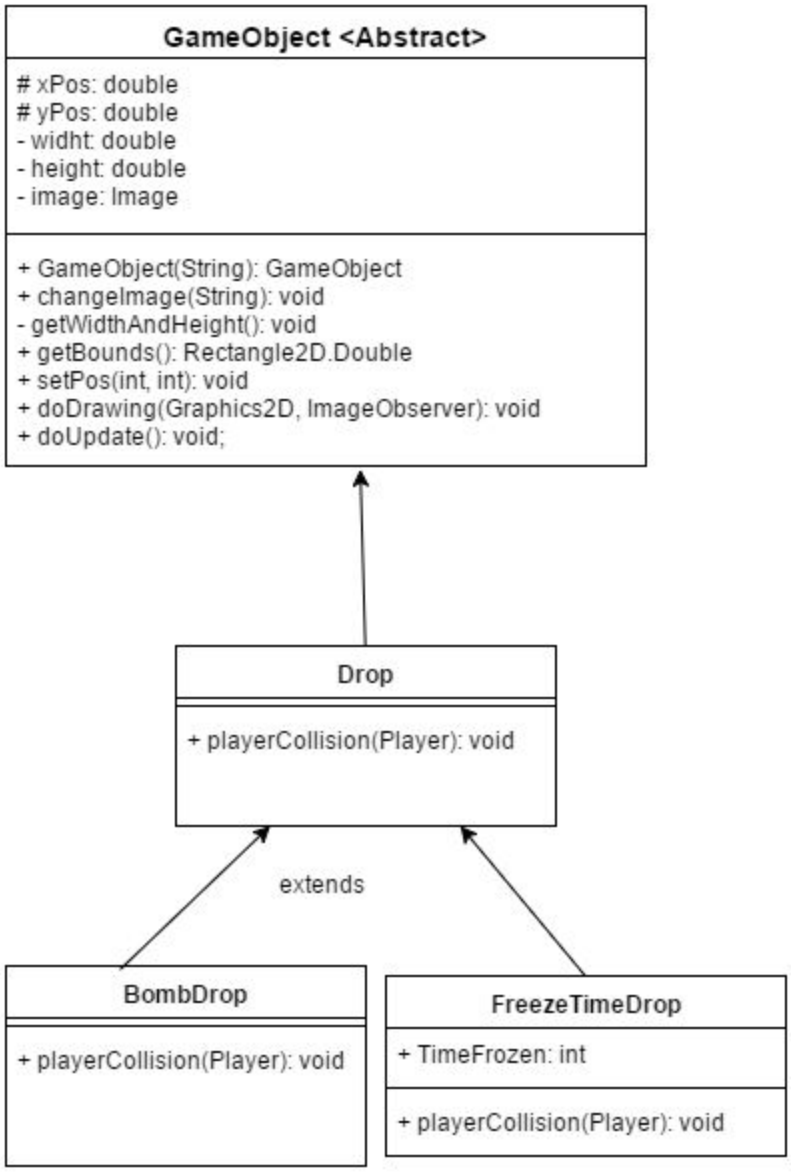
2. During the analysis and design phases of this extension use responsibility driven design and UML (push to the repository the single PDF file including all the documents produced) (8 pts)



This design used the decorator design pattern because it seemed like a good method to be able to stack different kind of effects on one drop if we wanted it. This ended up not working because the drop needed to be a GameObject too, otherwise it couldn't be drawn. If GameObject was or had an interface we could have extended GameObject in DropInterface. Rebuilding GameObject to have an interface would require a major rewrite which wasn't worth the trouble, especially because we didn't need it to use the decorator design. Instead we use normal inheritance where we overwrite playerCollision() and call the super.playerCollision().

Extending GameObject in DropDecorator would have caused an needlessly big constructor and would make the existence of Drop be useless.

We ended up with this design:



Note: Drop also includes many other methods,
these are left out because they were not the main focus of the design

In both designs the Extensions/Decorators were responsible for their own special feature e.g. BombDrop is responsible for exploding all the balls and calling super.playerCollision(player) to fire other features.
They don't need to share anything because Drop Handles all the other stuff.

Drop is responsible for adding points and movement,
Drop has to share nothing.
GameObject is responsible for the generic object behaviour, getting drawn/ providing collision boxes.
GameObject also has to share Position, width, height and Image.

Exercise 3:

1. Google asks its employees to spend 20% of their time at Google to a project that their job description does not cover. As a result of the 20% Project at Google, we now have Gmail, AdSense, and Google News, among the others.
This is your occasion to have similar freedom. You can decide what to do next to your game:5 It can be an extension/improvement from any perspective, such as improved code quality, or novel features.
Define your own requirements and get them approved by your teaching assistant. Afterwards you must implement the requirements. (22 pts).

Multiplayer
-Separate lives
-Extra revival powerup
-Individual power-ups
As a player i want to be able to play together with a friend on the same keyboard.

2. During the analysis and design phases of this extension use responsibility driven design and UML (push to the repository the single PDF file including all the documents produced) (8 pts).

To make implementing the multiplayer mode easier, we first changed the current methods the player uses to handle input and textures to a strategy design pattern. This way, we could make a second PlayerScheme fairly easily. Then the only thing left to do was change some code related to the creation of the Player object so that it creates two players (when in multiplayer mode) with two different PlayerSchemes, and change the levels to include two spawnpoints. The UML of the PlayerSchemes can be found in Exercise 1a.1. Some things for the player were handled globally, like the number of HookAndRopes shot by the player are now handled by the player object itself to separate this per player. The number of lives was also handled globally, but this is kept this way (at least for now). This increases the coop feeling as both player share their lives.