Assignment 7 - Designing The Game

Loic Konan

Description

To use **Object Oriented Programming** mindset to clarify what we need to implement, what classes are needed, how those classes are related and finally to specifies how they will carry out their resposibilities.

Dry (Don't Repeat Yourself)

- Don't write duplicate code.
- A class should do its own thing. If two classes are doing the same thing, to the same data, maybe it should be its own class entirely.
- Similarly, if you have a block of code in more than two places consider making it a separate method.

Single Responsibility Principle (SRP)

- A class should be written to handle one defined thing, and handle it well.
- The definition of "one" is the question here.
- Think along the lines of decoupling. For example when **ClassA** depends heavily on **ClassB**, the chances of **ClassA** being affected when **ClassB** is changed are high. We don't want this to happen.

Favor Composition over Inheritance

- To favor composition over inheritance is a design principle that gives the design higher flexibility.
- It is more natural to build classes out of various separate components rather than trying to find commonality between them in order to create an inheritance hierarchy.

Requirements / Attributes

When writing "requirements" for your classes (and we are following a very loose design process for now) you should think in the following terms:

- The **< *thing* >** should provide **< *something* >** so we can do **< *this* >**.

- They don't ALL have to fit this exactly, but each "requirement" or "attribute" should at least have a < ***subject*** > => < verb > approach. **Example:**

- A **score** (the **< thing >**) should know its **value** (the **< something >**) so it can be **displayed** (the **< this >**) on a game window.

- A **player** (the **< thing >**) should know its **location** (the **< something >**) so it can be **checked** (the **< this >**) for collisions.

- Without all the keywords embedded now:

    - A debris item should know its speed and direction so we can update its location.

  - A player should know its speed and direction so we can update its location. What do we notice about a debris item and a player? Seems to be a lot of overlap! We can leverage that information in our design!

Files

#	File	Description
1	Banner	Banner for Assignment
1	UntitledDiagram.png	Visual model of the classes using UML

Instructions

1. **Identify the classes and objects to be used in the program.**
2. **Define the attributes for each class.**
3. **Define the behaviors for each class.**
4. **Define the relationship between classes.**

Possible Classes

Player

- Has a Shape
- Has a Size
- Has a Color
- Has a Speed
- Has a Location (could change)
- Can move in any direction using keys
- Can collide with other "objects"

Debris

- Has a Shape
- Has a Size
- Has a Color
- Has a Speed
- Has a Location (could change)
- Can move in any direction
- Can collide with other "objects"

Scoring

- When a Player comes collides with Debris score is negatively effected.
- When a piece of Debris leaves game screen (on the left), score is positively effected.

Text

- Has a Font (can change)
- Has a Location
- Has a Color

- Has a Size

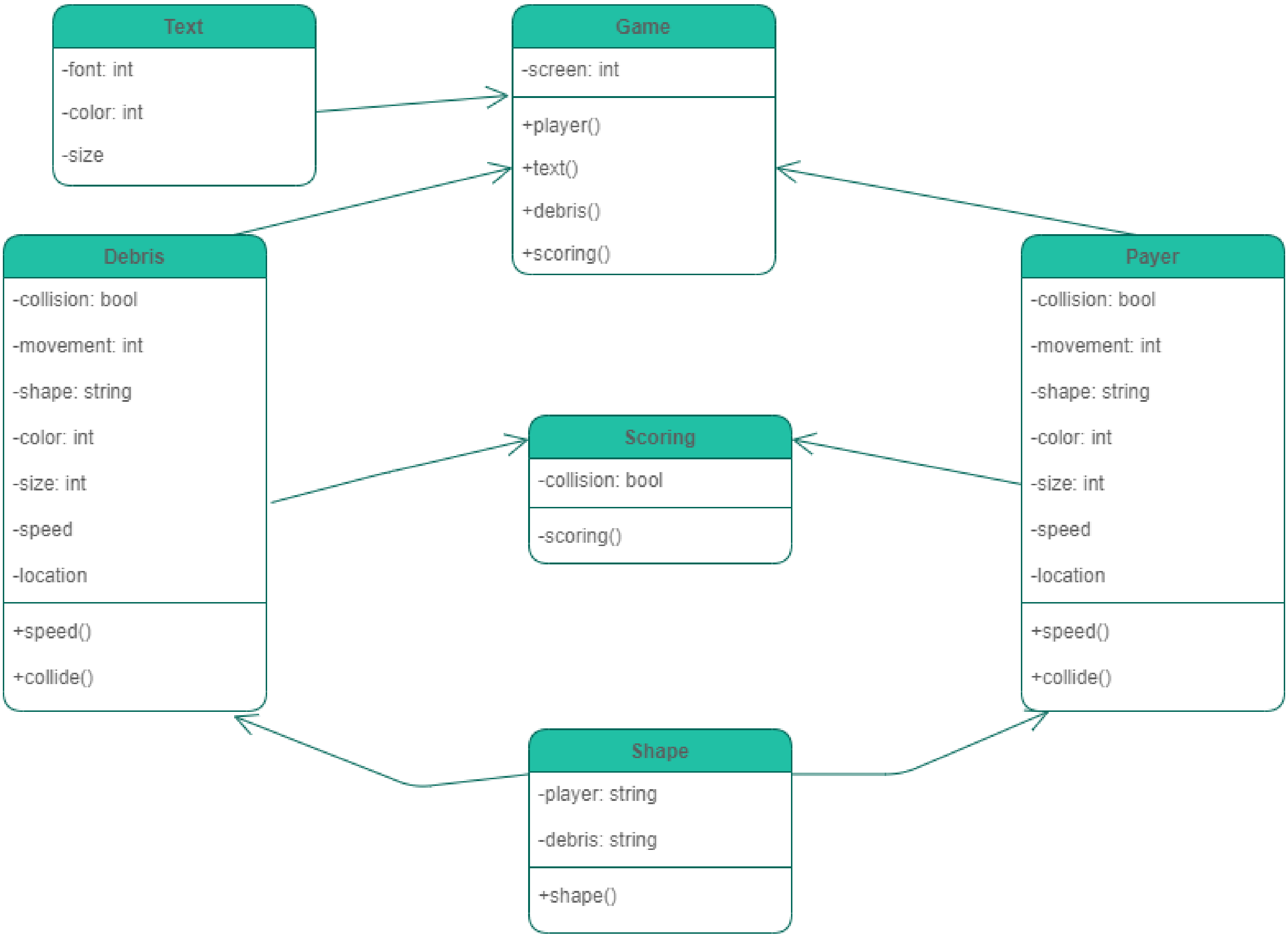
Game

- Has player(s)
- Has score(s)
- Has debris(s) (yes "debris" can be plural but it doesn't make the point)

Shape

- Is a player
- Is a debris

Diagram


Text
-font: int
-color: int
-size
Game
-screen: int
+player()
+text()
+debris()
+scoring()
Debris
-collision: bool
-movement: int
-shape: string
-color: int
-size: int
-speed
-location
+speed()
+collide()
Scoring
-collision: bool
-scoring()
Payer
-collision: bool
-movement: int
-shape: string
-color: int
-size: int
-speed
-location
+speed()
+collide()
Shape
-player: string
-debris: string
+shape()