Game Screen Elements Subsystem

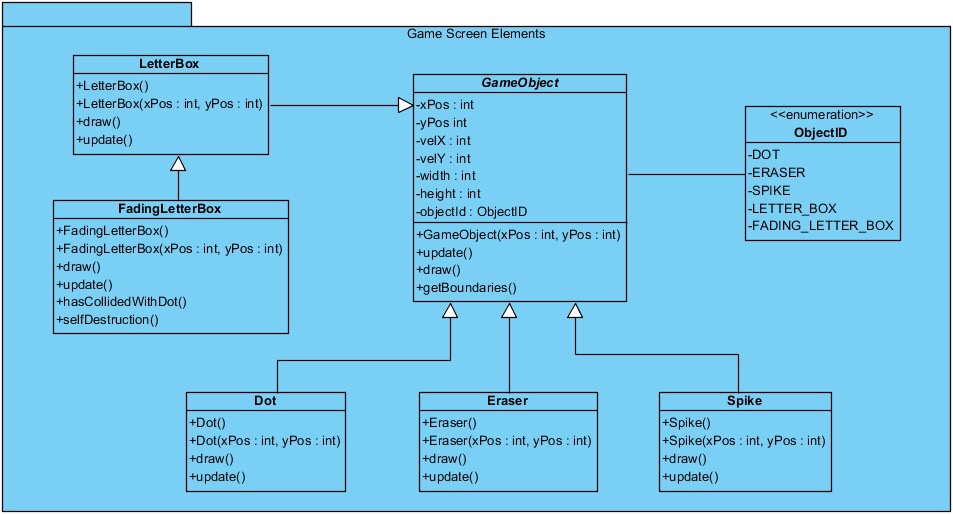
“Game Screen Elements Subsytem” plays a key role in declaring the objects which are shown in the screen while the game starts to running. There are many significant objects including “FadingLetterBox”, “LetterBox”, “Dot”, “Eraser”, “Spike” and “GameObject”. This subsystem has also Enumaration named as “ObjectID” to indicate the types of different objects.

Figure 1 – Package Diagram of GameScreenElements

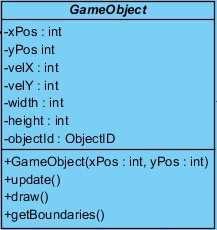
***GameObject* Class**

Figure 13 - GameObject Class

“GameObject” is an essential class for this subsystem. All of the objects have to keep an account of position information, velocity information, size information and object type hence, they use “GameObject” abstract class as a parent class which has all these attributes. This class stores the object type as “objectId” by using enumaration. Each object has a unique “objectId” defined as enum type. Also, there are abstract methods including “*update()”*, “*draw()”* and *“getBoundaries()”* since all objects may act differently to these operations. “GameObject” will be instantiated whenever the user attemts to play the game.

A short description of the attributes and the abstract methods in the GameObject class to clarify the functionality of each attribute and method:

**Attributes:**

* **xPos :** The x coordinate of object’s position.
* **yPos :** The y coordinate of object’s position.
* **velX :** The velociy of object in the direction of x coordinate.
* **velY :** The velociy of object in the direction of y coordinate.
* **width :** The width of object.
* **height :** The height of object.
* **objectId :** The enum type of object.

**Constructor:**

* **GameObject(xPos : int, yPos: int) :** This will initialize a game object according to given position values.

**Methods:**

* **draw() :** This is an abstarct method. All objects will include their own implementations but generally it will achieve drawing the object according to its size and position information.
* **update() :** This is also an abstract method. It will manage the position of object accoridng to its updated position information.
* **getBoundaries() :** This is also an abstract method and will be responsible for returning the boundaries of an object according to some attributes such as position, width and height.

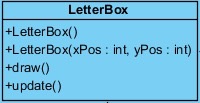
***LetterBox* Class**

Figure 3 - LetterBox Class

The “LetterBox” class has the same attributes as the GameObject class but also includes two constructors, the implementations of *“draw()”* and “*update()*” abstract methods. This class will manage the game platform by drawing a number of letters and creating different sentences. The player has to jump over these letters without dropping between them. The LetterBox has a child class representing the fading letters which will be defined below.

The description of abstract methods is given above.

**Constructors:**

* **LetterBox() :** This is the default constructor.
* **LetterBox(xPos : int, yPos : int ) :** This is the constructor where the position information will be given, thus the letter will be instantiated according to “xPos” and “yPos” values.

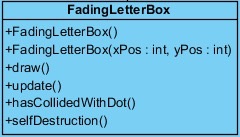
***FadingLetterBox* Class**

Figure 3 - LetterBox Class

The “FadingLetterBox” class is a child class of “LetterBox” class and has the same attributes as its parent class but also includes “*hasCollidedWithDot()*”and “*selfDestruction()*” methods as well as the implementations of *“draw()”* and “*update()*” abstract methods. This class will be initialized after the level including fading letters is played. The FadingLetterBox object has the abilty to disappear when the collision with the Dot object is detected by the “*hasCollidedWithDot()*” method. The “*selfDestruction()*” method enable the objects of this class to fade away in case of collision.

**Constructors:**

* **FadingLetterBox() :** This is the default constructor.
* **FadingLetterBox(xPos : int, yBox : int) :** This is the constructor where the position information will be given, thus the fading letter will be instantiated according to “xPos” and “yPos” values.

**Methods:**

* **hasCollidedWithDot() :** This method will detect the time when the Dot has collided a letter then, the letter will start to fade away.
* **selfDestruction() :** This method will enable the objects of this class to fade away in case of collision detected.

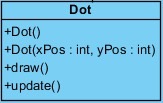
***Dot* Class**

Figure 4 - Dot Class

The “Dot” class has the same attibutes as the GameObject class and the implementations of *“draw()”* and “*update()*” abstract methods exist in the class. The Dot object is as essential class for the game since it stands for the player who aims to finish the levels successfully. It will be created when the player starts to play first level.

**Constructors:**

* **Dot () :** This is the default constructor.
* **Dot (xPos : int, yBox : int) :** This is the constructor where the position information will be given, thus the Dot will be instantiated according to “xPos” and “yPos” values.

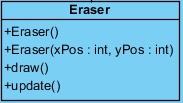
***Eraser* Class**

Figure 5 - Eraser Class

The “Eraser” class will be initialized when the player manages to play the level including the Eraser objects aimed at droping from top of the screen and hitting the Dot object as player. Hence, the Eraser object will take a part in exposing the player to some obstacles during the game.

**Constructors:**

* **Eraser() :** This is the default constructor.
* **Eraser(xPos : int, yBox : int) :** This is the constructor where the position information will be given, thus the Eraser will be instantiated according to “xPos” and “yPos” values.

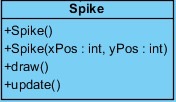
***Spike* Class**

Figure 6 - Spike Class

The “Spike” class stands for another obstacle type during the game. It has the same attributes as the GameObject. The Spike object will be positioned according to its attributes including “xPos” and “yPos”.

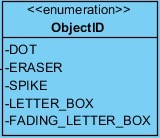
**Enumaration (ObjectID)**

Figure 7 – ObjectID as Enum Types

“Game Screen Elements Subsytem” includes a lot of object types such as “Dot”, “Eraser”, “Spike”, “LetterBox” and “FadingLetterBox”. In attempt to make our implementation organization easy, we use enumaration types by specifying “ObjectID” related to each unique object. For instance, we can call “objectId.DOT” as objectId attribute of the Dot object when we define it in the “ObjectID” enumaration only once, since all objects use the GameObject as parent class having “objectID” attribute. With the advantage of enum types, the GameObject class can use different objects by calling only their unique “objectId”.