**3.4. Game Logic Subsystem Interface**

Game Logic subsystem will be the bottom layer in three-tier system which will contain the key components of the product. Key class in this subsystem is GameObject class which is an abstract class for nearly every class in following subsystem. GameObject class is the keystone for 10 classes in the subsystem. Moreover, there is Room class and GameMap class for arranging each room component inside the game map. There are also compositions between GameMap – Room, GameMap – GameObject and GameObject – Sprite classes.

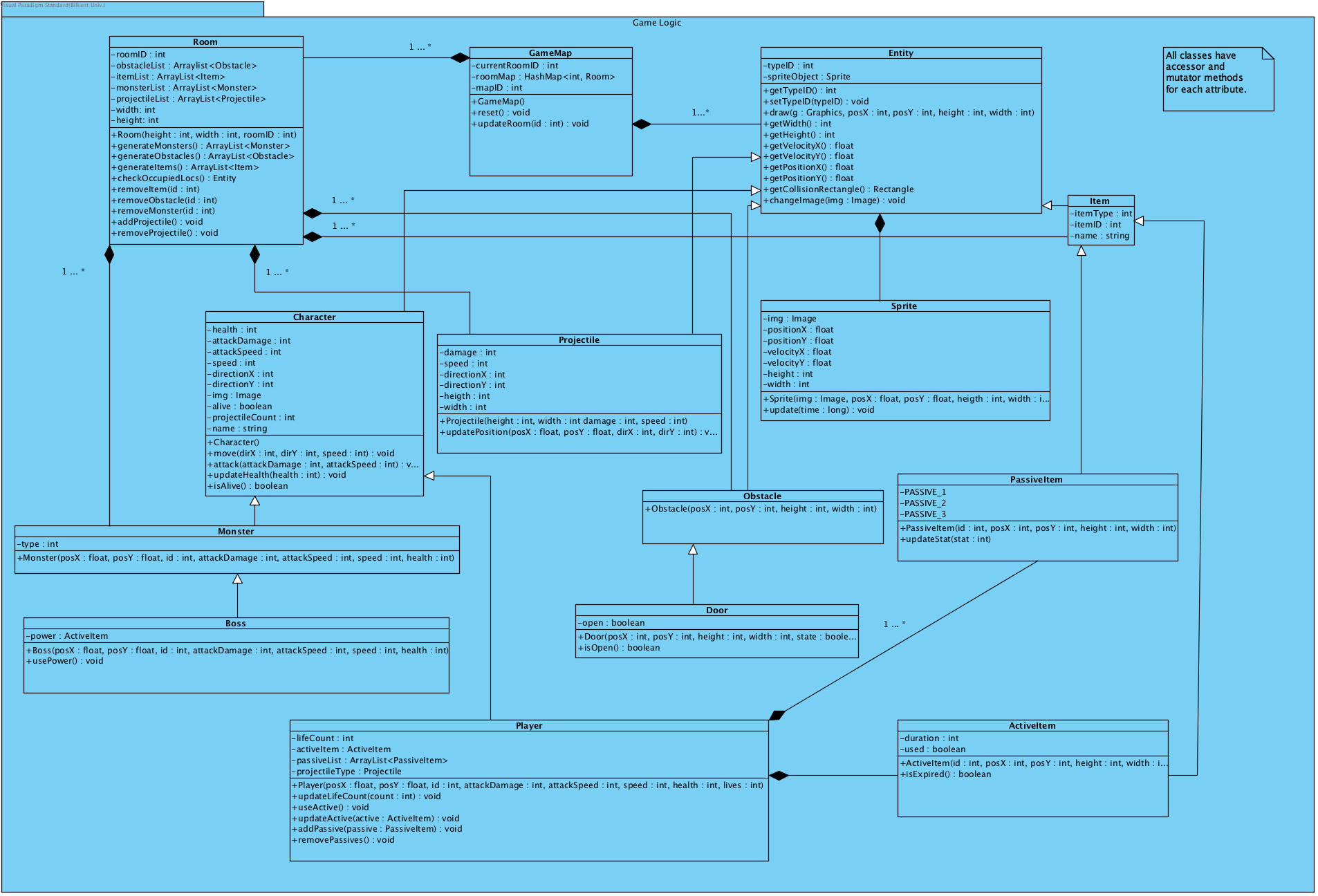
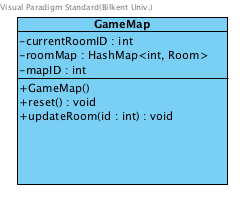


Figure ...

****

**GameMap Class**

🡪 This class is the top class of the Game Entities subsystem; it contains all Room objects which essentially generate every core component of the game excluding Player object. In other words, GameMap handles every level in the game.

***Attributes:***

**private int currentRoomID:** This attribute holds an integer to represent a roomID to determine which room is currently played or used when calling updateRoom method.

**private HashMap<int, Room> roomMap:**

**private int MapID:**

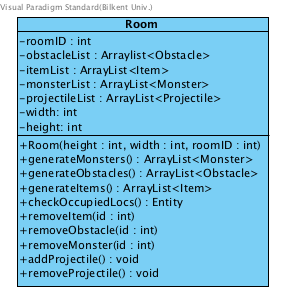
***Constructor:***

**GameMap ():** Default constructor with default attributes.

***Methods:***

**public void reset():**

**public void updateRoom (int id):**

****

**Room Class**

🡪 Room class contains all the components of the game; all rooms are initialized from the start of the game with Monster, Obstacle and Item objects. This class also handles boundary problems, checks for occupied locations in each room and generate objects accordingly. Each room has a roomID to select the target room accordingly.

***Attributes:***

**private int roomID:**

**private ArrayList<Obstacle> obstacleList:**

**private ArrayList<Item> itemList:**

**private ArrayList<Monster> monsterList:**

**private ArrayList<Projectile> projectileList:**

**private int width:**

**private int height:**

***Constructor:***

**Room (int height, int width, int roomID):**

***Methods:***

**public ArrayList<Monster> generateMonsters ():**

**public ArrayList<Obstacle> generateObstacles ():**

**public ArrayList<Item> generateItems ():**

**public Entity checkOccupiedLocs():**

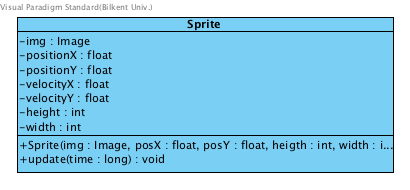
**public void removeItem (int id):**

**public void removeObstacle (int id):**

**public void removeMonster (int id):**

**public void addProjectile ():**

**public void removeProjectile ():**

****

**Sprite Class**

**🡪** Sprite class’ purpose is to update the position of Sprite object which contains an Image object inside. It has a method called “update” which manipulates the position of Sprite object in a given time interval by multiplying velocity values with given time interval to produce new positions.

***Attributes:***

**private Image img:** Image object for the appearance of Sprite object which will be manipulated over time.

**private float positionX:** x coordinate of the top left corner of object.

**private float positionY:** y coordinate of the top left corner of object.

**private float velocityX:** x coordinate of the velocity of object.

**private float velocityY:** y coordinate of the velocity of object.

**private int height:** height of the object.

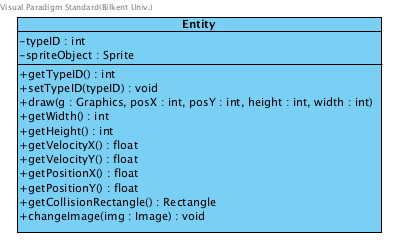
**private int width:** width of the object.

***Constructors:***

**public Sprite (Image img, float posX, float posY, int height, int width):** Constructor for Sprite object with given attributes.

***Methods:***

**public void update (long time):** This method takes a time parameter and updates Sprite object’s position with respect to time parameter. Sprite object’s velocity attributes are multiplied with time parameter and added to position parameters to determine new location.

**Entity Class**

**🡪** Entity is the abstract class and key class for all game objects. Every class from this point on essentially extends this abstract class, i.e. contain this class’ attributes and can use this it’s methods for core transitions in game. Entity also contains a Sprite object to manipulate in game appereance.

***Attributes:***

**private int typeID:** This attribute is for child classes to determine which type of Entity the object is.

**private Sprite spriteObject:** Sprite instance for enabling movement.

***Methods:***

**public int getTypeID():** Accessor for the typeID attribute.

**public int setTypeID (int typeID):** Mutator for the typeID attribute.

**public void draw (Graphics g, int posX, int posY, int height, int width):** This method is to draw Entity itself with given positions for upper left corner and height, width. To enable drawing method also has a Graphics object to enable drawing.

**public int getWidth():** Method for returning the width of the image from the game object, which is inside spriteObject attribute.

**public int getHeight():** Method for returning the height of the image from the game object, which is inside spriteObject attribute.

**public float getVelocityX():** Method for returning the x-component of velocity of the image from the game object, which is inside spriteObject attribute.

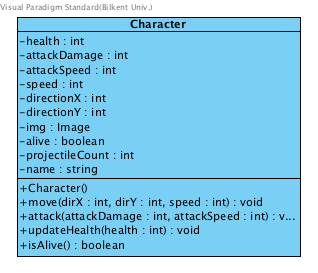
**public float getVelocityY():** Method for returning the y-component of velocity of the image from the game object, which is inside spriteObject attribute.

**public float getPositionX():** Method for returning the x-component of upper left corner’s position of the image from the game object, which is inside spriteObject attribute.

**public float getPositionX ():** Method for returning the y-component of upper left corner’s position of the image from the game object, which is inside spriteObject attribute.

**public Rectangle getCollisionRectangle():** Method forreturning the boundarie of spriteObject’s image attribute.

**public void changeImage(Image img):** Method for replacing the Entity’s image.

****

**Character Class**

***Attributes:***

**private int health:** Attribute to indicate a character’s health.

**private int attackDamage:** Attribute to indicate a character’s attack damage.

**private int attackSpeed:** Attribute to indicate a character’s attack speed.

**private int speed:** Attribute to indicate a character’s move speed.

**private int directionX:** Attribute to indicate a character’s movement direction’s x-component.

**private int directionY:** Attribute to indicate a character’s movement direction’s y-component.

**private Image img:** Attribute to indicate a character’s health.

**private boolean alive:** Attribute to indicate if a character is alive or not.

**private int projectileCount:** Attribute to indicate how many projectiles does a character fire when attack() function is called.

**private String name:** Attribute to indicate character’s name.

***Constructor:***

**public Character ():** Default constructors with default attributes.

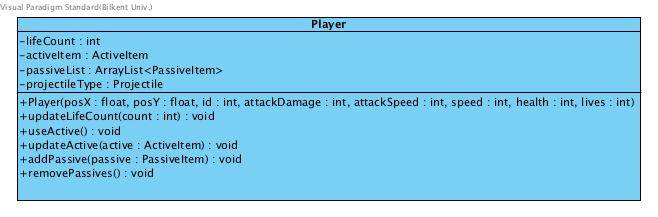
***Methods:***

**public void move (int dirX, int dirY, int speed):** This method takes 3 parameters; 2 int parameters which are x and y components to indicate directions to move to and an int parameter which is the speed attribute of Character object to determine how fast object going to move.

**public void attack (int attackDamage, int attackSpeed):** This method takes 2 int parameters; attackDamage attribute of Character object and attackSpeed attribute of Character object. When method called character while fire Prjoctile objects which will have the same parameters.

**public void updateHealth (int health):** This method is to update Character object’s health attribute in case of a decrease or increase in health while playing the game.

**public boolean isAlive ():** This method checks the alive attribute of Character object to determine if character is alive or not.

**Player Class**

***Attributes:***

**private int lifeCount:** This attribute will hold how many lives does Player object has.

**private ActiveItem activeItem:** Attribute to hold an ActiveItem object for Player object which it can use throughout the game when it posseses one. Initialized to null at the start.

**private ArrayList<PassiveItem> passiveList:** ArrayList attribute hold the PassiveItem objects of Player Object. Since passive items are permanent when Player gets it, there will be more than one PassiveItem objects of a Player object throughout the game.

**private Projectile projectileType:** Attribute for attack() method. This attribute will shape the projectile’s behavior.

***Constructor:***

**public Player (float posX, float posY, int id, int attackDamage, int attackSpeed, int speed, int health, int lives):** Constructor with parameters that come from Entity or Character class except lifeCount attribute.

***Methods:***

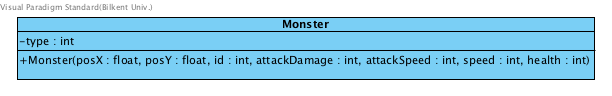
**public void updateLifeCount(int count):** Method for increasing or decreasing lifeCount attribute.

**public void useActive():** Method for using ActiveItem object that is inside Player object

**public void updateActive(ActiveItem active):** Method for either adding or deleting ActiveItem from Player object

**public void addPassive(PassiveItem passive):** Method foradding a PassiveItem object to PassiveItem ArrayList that is inside Player object.

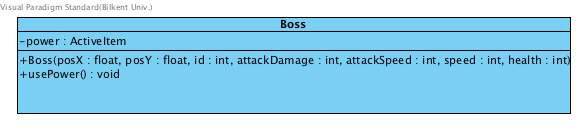
**public void removePassive():**Method for removing a PassiveItem object from PassiveItem ArrayList that is inside Player object.

**Monster Class**

***Attributes:***

**private int type:** attribute for determining the type of monster.

***Constructor:***

**Monster (float posX, float posY, int id, int attackDamage, int attackSpeed, int speed, int health):** Constructor with position, attack damage, attack speed, speed and health parameters that come from Character and Entity classes. Additionally, id parameter establishes type of monster.

**Boss Class**

***Attributes:***

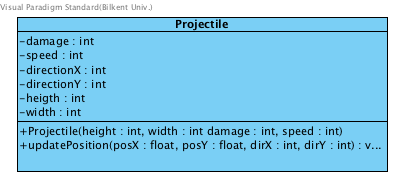
**private ActiveItem power:** This attribute holds the power of boss which is also an ActiveItem.

***Constructor:***

**Boss (float posX, float posY, int id, int attackDamage, int attackSpeed, int speed, int health):** Constructor with position, attack damage, attack speed, speed, id, health parameters that come from Monster, Character and Entity classes.

***Methods:***

**public void usePower():** This method enables Boss to use its power.

**Projectile Class**

***Attributes:***

**private int damage:** Attribute for holding the value of projectile damage.

**private int speed:** Attribute for holding the value of projectile’s movement speed.

**private int directionX:** Attribute to indicate projectile’s movement direction’s x-component.

**private int directionY:** Attribute to indicate projectile’s movement direction’s y-component.

**private int height:** Attribute for holding projectile’s image height.

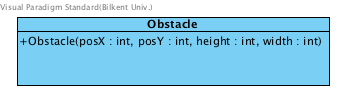
**private int width:** Attribute for holding projectile’s image width.

***Constructor:***

**public Projectile (int heigth, int width, int damage, int speed):** Constructor with size, damage and speed.

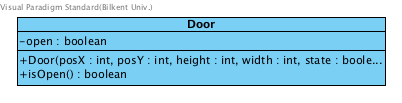
***Methods:***

**public void updatePosition(float posX, float posY, int dirX, int dirY, int speed):** This method takes position of left upper corner as two parameters and updates Sprite object’s position with respect to direction parameters.

**Obstacle Class**

***Constructor:***

**public Obstacle (int posX, int posY, int height, int width):** Constructor with position and height, width attributes. Position and height - width attributes come from spriteObject which is inside Entity class.

**Door Class**

***Attributes:***

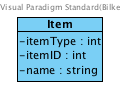
**private boolean open:** Boolean attribute to indicate the state of the door, initialized false at first.

***Constructor:***

**Door(int posX, int posY, int height, int width, boolean state):** Constructor with position and height, width attributes and the boolean state. Position and height, width attributes comes from spriteObject which is inside Entity class.

***Methods:***

**public boolean isOpen():** Method for checking the open attribute of Door object to determine if a room is completed and Player has been granted to pass the room.

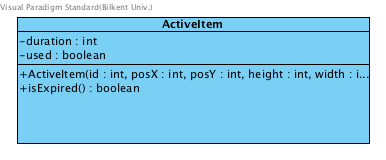
**Item Class**

***Attributes:***

**private int itemType:** Attribute to indicate itemType whether it is a PassiveItem or ActiveItem.

**private int itemID:** Attribute to indicate which item it is inside the PassiveItem list or ActiveItem list.

**private String name:** Attribute for the name of the Item object.

**ActiveItem Class**

***Attributes:***

**private int duration:** Attribute for indicating the expiration duration of ActiveItem

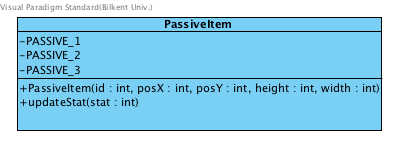
**private boolean used:** Boolean attribute to enable active item, initialized false when item acquired.

***Constructor:***

**ActiveItem (int id, int posX, int posY, int height, int width):** Constructor with an int parameter id to determine type. Position and height, width attributes come from spriteObject which is inside Entity class.

***Methods:***

**public boolean isExpired():** Method for checking the duration attribute of ActiveItem object once it came into use.

**PassiveItem Class**

***Attributes:***

**private constant int PASSIVE\_1:**

**private constant int PASSIVE\_2:**

**private constant int PASSIVE\_3:**

**🡪** These attributes are to determine types of passive items

***Constructor:***

**public PassiveItem (int id, int posX, int posY, int height, int width);** Constructor with id parameter to determine type. Position and height, width attributes come from spriteObject which is inside Entity class.

***Methods:***

**public void updateStat(int stat):** Method for adding the stat upgrade to player taking an int parameter to determine amount of upgrade.