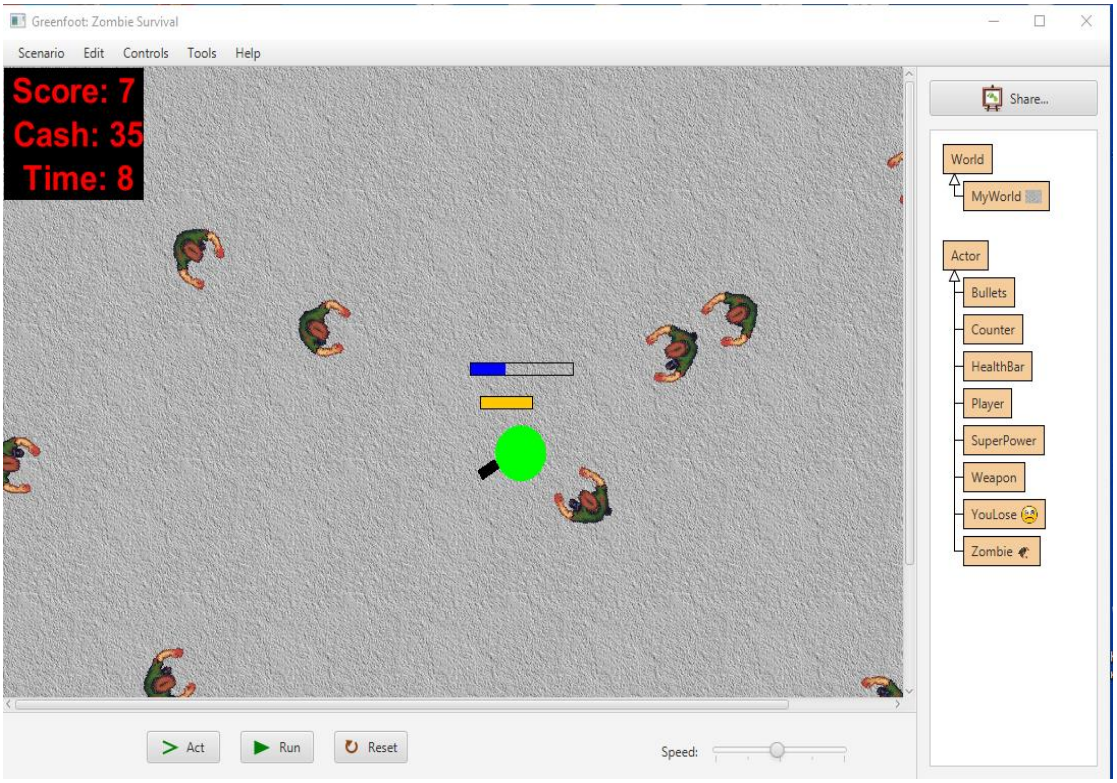
Άσκηση 10η

Δημιουργήστε τις κλάσεις και τις υποκλάσεις όπως απεικονίζονται παρακάτω, γράψτε τους κώδικες στους αντίστοιχους editor και «τρέξτε» το πρόγραμμα….



Οι φωτογραφίες που θα χρειαστούν για την δημιουργία του παιχνιδιού βρίσκονται στον παρακάτω σύνδεσμο…

https://drive.google.com/file/d/1SLim2icrrjPzKF7fdlj12AhzcKl1H3ov/view?usp=sharing

Ο παρακάτω κώδικας δημιουργεί έναν κόσμο στο παιχνίδι, συμπεριλαμβάνοντας τον παίκτη, έναν μετρητή, μια μπάρα υγείας, ένα όπλο και μια υπερδύναμη. Ο κώδικας διαχειρίζεται τον χρόνο του παιχνιδιού, την ταχύτητα εμφάνισης ζόμπι, και την εμφάνιση ζόμπι σε τυχαίες θέσεις στον κόσμο. Επίσης, υπάρχει έλεγχος για το πάτημα του πλήκτρου space και η ενημέρωση της μπάρας υγείας και του μετρητή.

```
import greenfoot.*;
/**
 * Write a description of class MyWorld here.
 *
 * Kaffetzis Vasileios
 * @version (a version number or a date)
 */
public class MyWorld extends World {
    int count = 0; // Μετρητής για τον χρόνο που έχει περάσει από την έναρξη του παιχνιδιού
    int spawnSpeed = 30; // Η ταχύτητα εμφάνισης ζόμπι θα γίνει πιο γρήγορη αν πάρει χαμηλότερη τιμή από το spawnSpeed
    int randomSpawn; // Μια μεταβλητή για τυχαίο αριθμό
    public Player mainPlayer = new Player(); // Ο κεντρικός παίκτης του παιχνιδιού
    Counter counter = new Counter(); // Ο μετρητής των πόντων του παίκτη
    HealthBar healthbar = new HealthBar(); // Η μπάρα υγείας του παίκτη
    Weapon weapon = new Weapon(counter); // Το όπλο του παίκτη
    SuperPower superPower = new SuperPower(); // Η υπερδύναμη του παίκτη

    /**
     * Κατασκευαστής της κλάσης MyWorld. Δημιουργεί τον κόσμο και τα αντικείμενα που περιέχει.
     */
    public MyWorld() {
        super(1000, 700, 1); // Δημιουργία κόσμου με διαστάσεις 1000x700 pixels
        mainPlayer = new Player(weapon, superPower);
        addObject(mainPlayer, getWidth() / 2, getHeight() / 2);
        addObject(counter, 70, 60);
        addObject(healthbar, mainPlayer.getX() - 5, mainPlayer.getY() - 50);
        addObject(weapon, 950, 30);
        addObject(superPower, mainPlayer.getX() + 10, mainPlayer.getY() - 80);
    }
    /**
     * Επιστρέφει τον κεντρικό παίκτη του παιχνιδιού.
     * @return Ο παίκτης του παιχνιδιού.
     */
    public Player getPlayer()
    {
        return mainPlayer;
    }
    /**
     * Η μέθοδος act καλείται αυτόματα κάθε frame και εκτελεί τις αναγκαίες ενέργειες του κόσμου.
     */
    public void act()
    {
        count++;
        if (count % 600 == 0)
        {
            spawnSpeed--; // Μειώνει την ταχύτητα εμφάνισης ζόμπι κάθε 600 frames
        }
        while (Greenfoot.isKeyDown("space"))
        {
            Greenfoot.delay(1); // Καθυστερεί το παιχνίδι όσο το πλήκτρο space είναι πατημένο
        }
        spawnZombies(); // Εκτελεί τη μέθοδο για την εμφάνιση ζόμπι
    }
    /**
     * Η μέθοδος spawnZombies ελέγχει αν είναι κατάλληλο να εμφανιστεί ένα ζόμπι και αν ναι, το τοποθετεί σε μια τυχαία θέση στον κόσμο.
     */
    public void spawnZombies ()
    {
        if (count % spawnSpeed == 0)
        {
        randomSpawn = Greenfoot.getRandomNumber(8);
            switch(randomSpawn)
        {
        case 0: addObject(new Zombie(mainPlayer, counter), 0, 0); break;
        case 1: addObject(new Zombie(mainPlayer, counter), getWidth() / 2, 0); break;
        case 2: addObject(new Zombie(mainPlayer, counter), getWidth(), 0); break;
        case 3: addObject(new Zombie(mainPlayer, counter), 0, getHeight() / 2); break;
        case 4: addObject(new Zombie(mainPlayer, counter), getWidth(), getHeight() / 2); break;
        case 5: addObject(new Zombie(mainPlayer, counter), 0, getHeight()); break;
        case 6: addObject(new Zombie(mainPlayer, counter), getWidth() / 2, getHeight()); break;
        case 7: addObject(new Zombie(mainPlayer, counter), getWidth(), getHeight()); break;
        }
        }
    }
}
```

Η κλάση Player υλοποιεί τον χαρακτήρα του παίκτη στο παιχνίδι. Ο παίκτης μπορεί να κινείται προς τις τέσσερις κατευθύνσεις χρησιμοποιώντας τα πλήκτρα κίνησης, ενώ η περιστροφή του προσανατολίζεται προς τη θέση του ποντικιού. Ο παίκτης επίσης μπορεί να εκτοξεύει βολές με το πάτημα του ποντικιού. Η κατεύθυνση και η ποσότητα των βολών εξαρτώνται από το επίπεδο αναβάθμισης του εκτοξευτή βολών. Ο παίκτης έχει επίσης τη δυνατότητα χρήσης υπερδύναμης, η οποία ενεργοποιείται όταν ο μετρητής της υπερδύναμης ξεπερνά την τιμή 99. Σε αυτήν την περίπτωση, ο παίκτης εκτοξεύει πολλαπλές βολές προς διάφορες κατευθύνσεις. Ο παίκτης χάνει το παιχνίδι αν έρθει σε επαφή με ένα ζόμπι.

```
import greenfoot.*;
/**
 * Η κλάση Player αντιπροσωπεύει τον παίκτη στο παιχνίδι.
 * Περιλαμβάνει λειτουργίες όπως η κίνηση, η περιστροφή, ο εκτοξευτής βολών και η χρήση υπερδύναμης.
 */
public class Player extends Actor
{
    int speed = 3; // Η ταχύτητα κίνησης του παίκτη
    int time = 0; // Ο συνολικός χρόνος παιχνιδιού
    Weapon weapon; // Ο εκτοξευτής βολών του παίκτη
    SuperPower superPower; // Η υπερδύναμη του παίκτη
    int superTimer; // Μετρητής χρόνου για τη χρήση της υπερδύναμης
    /**
     * Κατασκευαστής της κλάσης Player χωρίς παραμέτρους.
     */
    public Player()
    {
        // Δημιουργία εικόνας για τον παίκτη
        setImage(new GreenfootImage(70, 50));
        getImage().setColor(Color.GREEN);
        getImage().fillOval(0, 0, 50, 50);
        getImage().setColor(Color.BLACK);
        getImage().fillRect(50, 25, 70, 10);
    }
    /**
     * Κατασκευαστής της κλάσης Player με παραμέτρους Weapon και SuperPower.
     */
    public Player(Weapon weapon, SuperPower superPower)
    {
        this.superPower = superPower;
        this.weapon = weapon;
        // Δημιουργία εικόνας για τον παίκτη
        setImage(new GreenfootImage(70, 50));
        getImage().setColor(Color.GREEN);
        getImage().fillOval(0, 0, 50, 50);
        getImage().setColor(Color.BLACK);
        getImage().fillRect(50, 25, 70, 10);
    }
    /**
     * Η μέθοδος act καλείται αυτόματα κάθε frame και εκτελεί τις αναγκαίες ενέργειες του παίκτη.
     */
    public void act()
    {
        time++;
        turnArround();
        moveArround();
        fireBullets();
        superPowerUsed();
        HitByZombie();
    }
    /**
     * Η μέθοδος turnArround προσανατολίζει τον παίκτη προς τη θέση του ποντικιού.
     */
    public void turnArround()
    {
        if (Greenfoot.getMouseInfo() != null)
            turnTowards(Greenfoot.getMouseInfo().getX(), Greenfoot.getMouseInfo().getY());
    }
    /**
     * Η μέθοδος moveArround διαχειρίζεται την κίνηση του παίκτη με τα πλήκτρα.
     */
    public void moveArround()
    {
        if (Greenfoot.isKeyDown("up"))
            setLocation(getX(), getY() - speed);
        if (Greenfoot.isKeyDown("left"))
            setLocation(getX() - speed, getY());
        if (Greenfoot.isKeyDown("down"))
            setLocation(getX(), getY() + speed);
        if (Greenfoot.isKeyDown("right"))
            setLocation(getX() + speed, getY());
    }
```

```
    /**
     * Η μέθοδος fireBullets εκτοξεύει βολές ανάλογα με το επίπεδο αναβάθμισης του εκτοξευτή βολών.
     */
    public void fireBullets()
    {
        if (Greenfoot.mousePressed(null) && weapon.weaponUpgrade == 1)
        {
            Bullets bullets = new Bullets();
            getWorld().addObject(bullets, getX(), getY());
            bullets.setRotation(getRotation());
            bullets.move(25);
        }
        else if (Greenfoot.mousePressed(null) && weapon.weaponUpgrade == 2)
        {
            Bullets bullets = new Bullets();
            getWorld().addObject(bullets, getX(), getY());
            bullets.setRotation(getRotation() - 10);
            bullets.move(20);
            Bullets bullets2 = new Bullets();
            getWorld().addObject(bullets2, getX(), getY());
            bullets2.setRotation(getRotation() + 10);
            bullets2.move(20);
        }
        else if (Greenfoot.mousePressed(null) && weapon.weaponUpgrade == 3)
        {
            Bullets bullets = new Bullets();
            getWorld().addObject(bullets, getX(), getY());
            bullets.setRotation(getRotation() - 10);
            bullets.move(20);
            Bullets bullets2 = new Bullets();
            getWorld().addObject(bullets2, getX(), getY());
            bullets2.setRotation(getRotation() + 10);
            bullets2.move(20);
            Bullets bullets3 = new Bullets();
            getWorld().addObject(bullets3, getX(), getY());
            bullets3.setRotation(getRotation());
            bullets3.move(20);
        }
    }
    /**
     * Η μέθοδος superPowerUsed χρησιμοποιεί την υπερδύναμη όταν ο μετρητής της είναι μεγαλύτερος από 99.
     * Ο χρήστης εκτοξεύει βολές προς πολλές κατευθύνσεις.
     */
    public void superPowerUsed()
    {
        if (superPower.superCount > 99 && superTimer < 35) //
        {
            Bullets bullets = new Bullets();
            getWorld().addObject(bullets, getX(), getY());
            bullets.setRotation(getRotation());
            bullets.move(20);
            Bullets bullets2 = new Bullets();
            getWorld().addObject(bullets2, getX(), getY());
            bullets2.setRotation(getRotation() - 60);
            bullets2.move(20);
            Bullets bullets3 = new Bullets();
            getWorld().addObject(bullets3, getX(), getY());
            bullets3.setRotation(getRotation() + 60);
            bullets3.move(20);
            Bullets bullets4 = new Bullets();
            getWorld().addObject(bullets4, getX(), getY());
            bullets4.setRotation(getRotation() - 100);
            bullets4.move(20);
            Bullets bullets5 = new Bullets();
            getWorld().addObject(bullets5, getX(), getY());
            bullets5.setRotation(getRotation() + 120);
            bullets5.move(20);
            Bullets bullets6 = new Bullets();
            getWorld().addObject(bullets6, getX(), getY());
            bullets6.setRotation(getRotation() - 120);
            bullets6.move(20);
            superTimer++;
        }
        if (superPower.superCount > 34)
        {
            superPower.superCount = 0;
            superTimer = 0;
        }
    }
    /**
     * Η μέθοδος HitByZombie ελέγχει αν ο παίκτης έχει επαφή με ένα ζόμπι.
     * @return Επιστρέφει true αν ο παίκτης έχει χτυπηθεί από ζόμπι, αλλιώς false.
     */
    public boolean HitByZombie()
    {
        Actor zombie = getOneObjectAtOffset(0, 0, Zombie.class);
        if (zombie != null)
        {
            return true;
            //getWorld().showText("You Lose! - You leasted" + (time/60) + "seconds", getWorld().getWidth() / 2, getWorld().getHeight() / 2
            //Greenfoot.stop();
        } else
            return false;
    }
}
```

Η κλάση Zombie υλοποιεί τη συμπεριφορά των ζόμπι στο παιχνίδι. Κατά τη δημιουργία ενός νέου ζόμπι, του ορίζεται μια εικόνα για την αρχική κατάστασή του. Κατά τη διάρκεια κάθε frame, το ζόμπι κινείται προς την κατεύθυνση του παίκτη, ενώ ταυτόχρονα εκτελείται μια διαδικασία αναπαραγωγής (animation) που αλλάζει την εικόνα του με κάθε frame, προκειμένου να δημιουργηθεί η εντύπωση κίνησης. Το ζόμπι επίσης ελέγχει αν συγκρούεται με βολές, μειώνοντας την υγεία του. Εάν η υγεία του ζόμπι φτάσει στο μηδέν, τότε ο παίκτης κερδίζει πόντους και χρήματα, και το ζόμπι αφαιρείται από τον κόσμο.

```
import greenfoot.*;
/**
 * Η κλάση Zombie υλοποιεί τη συμπεριφορά των ζόμπι στο παιχνίδι.
 */
public class Zombie extends Actor
{
    int frame = 0;
    int animateImage = 0;
    int animateSpeed = 5; // Προσδιορισμός υψηλότερης ταχύτητας αναπαραγωγής
    int count;
    int health = 2;
    Player player;
    Counter counter;
    int time;
    /**
     * Ο κατασκευαστής της κλάσης ορίζει την αρχική εικόνα του ζόμπι και αρχικοποιεί τις μεταβλητές.
     */
    public Zombie(Player mainPlayer, Counter counter)
    {
        this.player = mainPlayer;
        this.counter = counter;
        setImage("skeleton-idle_16.png");
        getImage().scale(80, 80);
    }
    /**
     * Η μέθοδος act καλείται κάθε frame και εκτελεί την αναπαραγωγή, την κίνηση και τον έλεγχο σύγκρουσης με βολές.
     */
    public void act()
    {
        count++;
        animate();
        moveAround();
        hitByBullets();
    }
    /**
     * Η μέθοδος setCounter ορίζει τον μετρητή χρημάτων για το ζόμπι.
     */
    public void setCounter(Counter counter)
    {
        this.counter = counter;
    }
    /**
     * Η μέθοδος animate εκτελεί την αναπαραγωγή του ζόμπι, αλλάζοντας την εικόνα του κάθε frame.
     */
    public void animate()
    {
        if (count % animateSpeed == 0)
        {
            if (animateImage > 16)
            {
                animateImage = 0;
            }
            setImage("skeleton-move_" + animateImage + ".png");
            animateImage++;
            getImage().scale(80, 80);
        }
    }
    /**
     * Η μέθοδος moveAround κινεί το ζόμπι προς τον παίκτη.
     */
    public void moveAround()
    {
        move(1);
        turnTowards(player.getX(), player.getY());
    }
```

```
    /**
     * Η μέθοδος hitByBullets ελέγχει αν το ζόμπι συγκρούεται με βολές, μειώνοντας την υγεία του αντίστοιχα.
     */
    public void hitByBullets()
    {
        Actor bullets = getOneIntersectingObject(Bullets.class);
        if (bullets != null)
        {
            health--;
            getWorld().removeObject(bullets);
        }
        if (health == 0)
        {
            counter.score++;
            counter.money += 5;
            getWorld().removeObject(this);
        }
    }
}
```

Η κλάση Bullets υλοποιεί τη συμπεριφορά των βολών στο παιχνίδι. Κατά την δημιουργία μιας νέας βολής, ορίζεται η εικόνα της ως ένα οριζόντιο κόκκινο ορθογώνιο διαστάσεων 10x2 pixel. Κατά τη διάρκεια κάθε frame, η βολή κινείται προς τα εμπρός κατά 10 pixel. Εάν η βολή φτάσει στα όρια του κόσμου, αφαιρείται από τον κόσμο για να μην καταναλώνει περισσότερους πόρους.

```
import greenfoot.*;
/**
 * Η κλάση Bullets υλοποιεί τη συμπεριφορά των βολών στο παιχνίδι.
 */
public class Bullets extends Actor
{
    /**
     * Ο κατασκευαστής της κλάσης αρχικοποιεί την εικόνα της βολής ως ένα οριζόντιο κόκκινο ορθογώνιο.
     */
    public Bullets()
    {
        setImage(new GreenfootImage(10, 2));
        getImage().setColor(Color.RED);
        getImage().fillRect(0, 0, 10, 2);
    }
    /**
     * Η μέθοδος act καλείται κάθε frame και κινεί τη βολή προς τα εμπρός κατά 10 pixel.
     * Εάν η βολή φτάσει στα όρια του κόσμου, αφαιρείται από τον κόσμο.
     */
    public void act()
    {
        move(10);
        if (isAtEdge())
        {
            getWorld().removeObject(this);
        }
    }
}
```

Η κλάση Weapon αντιπροσωπεύει το όπλο που μπορεί ο παίκτης να αναβαθμίσει στο παιχνίδι. Κατά τη δημιουργία ενός νέου αντικειμένου όπλου, ορίζεται η εικόνα του με ένα κείμενο που δείχνει τη λειτουργία του όπλου και το επίπεδο αναβάθμισής του. Κατά τη διάρκεια κάθε frame, ελέγχεται εάν ο παίκτης πατάει το όπλο και έχει αρκετά χρήματα για να αναβαθμίσει το όπλο του. Εάν ικανοποιούνται αυτές οι συνθήκες, μειώνεται το ποσό των χρημάτων του παίκτη και αυξάνεται το επίπεδο αναβάθμισης του όπλου. Τέλος, ελέγχεται εάν το επίπεδο αναβάθμισης έχει φτάσει το μέγιστο (3) και, αν ναι, διατηρείται στο μέγιστο.

```java
import greenfoot.*;
/**
 * Η κλάση Weapon αντιπροσωπεύει το όπλο που μπορεί ο παίκτης να αναβαθμίσει στο παιχνίδι.
 */
public class Weapon extends Actor
{
    Counter counter; // Ο μετρητής χρημάτων του παίκτη
    int weaponUpgrade = 1; // Το επίπεδο αναβάθμισης του όπλου
    /**
     * Ο κατασκευαστής της κλάσης αρχικοποιεί τον μετρητή χρημάτων και την εικόνα του όπλου.
     */
    public Weapon(Counter counter)
    {
        this.counter = counter;
        setImage(new GreenfootImage("Weapon \n Upgrade", 25, Color.BLACK, new Color(0, 0, 0, 0)));
    }
    /**
     * Η μέθοδος act εκτελείται κάθε frame και ελέγχει την αναβάθμιση του όπλου ανάλογα με τα χρήματα του παίκτη.
     */
    public void act()
    {
        // Έλεγχος αν ο παίκτης πατάει το όπλο και έχει αρκετά χρήματα
        if (counter != null && Greenfoot.mousePressed(this) && counter.money >= 50)
        {
            counter.money -= 50; // Μείωση των χρημάτων του παίκτη
            weaponUpgrade++; // Αύξηση του επιπέδου αναβάθμισης του όπλου
        }
        // Έλεγχος αν το επίπεδο αναβάθμισης ξεπερνά το μέγιστο
        if (weaponUpgrade > 3)
        {
            weaponUpgrade = 3;
        }
    }
}
```

Η κλάση SuperPower αναπαριστά τη δυνατότητα υπερφυσικής ενέργειας στο παιχνίδι. Κατά τη δημιουργία ενός νέου αντικειμένου υπερφυσικής ενέργειας, ορίζεται η αρχική εικόνα της με ένα μπλε γραφικό που αντιπροσωπεύει τη γραμμή υπερφυσικής ενέργειας. Κατά τη διάρκεια κάθε frame, η εικόνα ανανεώνεται για να αντικατοπτρίζει την τρέχουσα ποσότητα υπερφυσικής ενέργειας. Το αντικείμενο επίσης μετακινείται στη θέση του παίκτη και η μέθοδος useSuper αυξάνει την υπερφυσική ενέργεια κατά κάποιο ποσοστό κάθε 5 frames.

```java
import greenfoot.*;
/**
 * Η κλάση SuperPower αναπαριστά τη δυνατότητα υπερφυσικής ενέργειας στο παιχνίδι.
 */
public class SuperPower extends Actor
{
    final int SUPER_POWER_LIMIT = 100; // Το μέγιστο όριο υπερφυσικής ενέργειας
    int superCount; // Η τρέχουσα ποσότητα υπερφυσικής ενέργειας
    int count; // Μετρητής frames
    /**
     * Ο κατασκευαστής της κλάσης ορίζει την αρχική εικόνα της υπερφυσικής ενέργειας.
     */
    public SuperPower()
    {
        setImage(new GreenfootImage(SUPER_POWER_LIMIT + 2, 12));
        getImage().drawRect(0, 0, SUPER_POWER_LIMIT + 1, 11);
        getImage().setColor(Color.BLUE);
        getImage().fillRect(1, 1, superCount, 10);
    }
    /**
     * Η μέθοδος act καλείται κάθε frame και ανανεώνει την εικόνα της υπερφυσικής ενέργειας,
     * τοποθετεί το αντικείμενο στη θέση του παίκτη και εκτελεί τη μέθοδο useSuper.
     */
    public void act()
    {
        setImage(new GreenfootImage(SUPER_POWER_LIMIT + 2, 12));
        getImage().drawRect(0, 0, SUPER_POWER_LIMIT + 1, 11);
        getImage().setColor(Color.BLUE);
        getImage().fillRect(1, 1, superCount, 10);
        World world = getWorld();
        MyWorld myWorld = (MyWorld) world;
        setLocation(myWorld.getPlayer().getX() + 10, myWorld.getPlayer().getY() - 80);
        useSuper();
    }
```

```
    /**
     * Η μέθοδος useSuper αυξάνει την ποσότητα υπερφυσικής ενέργειας κάθε 5 frames.
     * Εάν η ποσότητα ξεπεράσει το μέγιστο, επαναφέρεται στο μηδέν.
     */
    public void useSuper()
    {
        count++;
        if (count % 5 == 0)
        {
            superCount++;
        }
        if (superCount > SUPER_POWER_LIMIT)
        {
            superCount = 0;
        }
    }
}
```

Η κλάση Counter υλοποιεί ένα αντικείμενο μετρητή που παρουσιάζει το σκορ, τα χρήματα και τον χρόνο στο παιχνίδι. Κατά τη δημιουργία ενός νέου μετρητή, ορίζεται η αρχική εικόνα του με τα αρχικά τιμήματα του σκορ, των χρημάτων και του χρόνου. Κατά τη διάρκεια κάθε frame, η εικόνα του μετρητή ανανεώνεται για να αντικατοπτρίζει τις τρέχουσες τιμές του σκορ, των χρημάτων και του χρόνου.

```
import greenfoot.*;
/**
 * Η κλάση Counter υλοποιεί ένα αντικείμενο μετρητή που παρουσιάζει το σκορ, τα χρήματα και τον χρόνο στο παιχνίδι.
 */
public class Counter extends Actor
{
    int score; // Το σκορ του παίκτη
    int money; // Τα χρήματα του παίκτη
    int time; // Ο χρόνος που έχει περάσει
    /**
     * Ο κατασκευαστής της κλάσης ορίζει την αρχική εικόνα του μετρητή με τα αρχικά τιμήματα του σκορ, των χρημάτων και του χρόνου.
     */
    public Counter()
    {
        setImage(new GreenfootImage("Score: " + score + "\n Cash: "+ money + "\n Time: "+ time / 60, 40, Color.RED, new Color(0, 0, 0)));
    }
    /**
     * Η μέθοδος act καλείται κάθε frame και ανανεώνει την εικόνα του μετρητή για να αντικατοπτρίζει τις τρέχουσες τιμές.
     */
    public void act()
    {
        time++;
        setImage(new GreenfootImage("Score: " + score + "\n Cash: "+ money + "\n Time: "+ time / 60, 40, Color.RED, new Color(0, 0, 0)));
    }
}
```

Η κλάση HealthBar υλοποιεί ένα γραφικό αντικείμενο μπάρας υγείας που αντιπροσωπεύει την υγεία του παίκτη στο παιχνίδι. Κατά τη δημιουργία του αντικειμένου, ορίζεται η αρχική εικόνα της μπάρας υγείας με ένα πορτοκαλί γραφικό που αντιπροσωπεύει την αρχική υγεία. Κατά τη διάρκεια κάθε frame, η εικόνα της μπάρας υγείας ανανεώνεται για να αντικατοπτρίζει την τρέχουσα υγεία του παίκτη. Επίσης, το αντικείμενο τοποθετείται στη θέση του παίκτη, και η μέθοδος loseHealth ελέγχει εάν ο παίκτης πλήττεται από έναν ζόμπι, μειώνοντας την υγεία κατά ένα.

```java
import greenfoot.*;
/**
 * Η κλάση HealthBar υλοποιεί ένα γραφικό αντικείμενο μπάρας υγείας που αντιπροσωπεύει την υγεία του παίκτη στο παιχνίδι.
 */
public class HealthBar extends Actor
{
    int health = 50; // Η τρέχουσα υγεία του παίκτη
    double specificHealth = (double) health; // Μια μεταβλητή υγείας με διπλή ακρίβεια
    /**
     * Ο κατασκευαστής της κλάσης ορίζει την αρχική εικόνα της μπάρας υγείας με ένα πορτοκαλί γραφικό που αντιπροσωπεύει την αρχική υγεία.
     */
    public HealthBar()
    {
        setImage(new GreenfootImage(52, 12));
        getImage().drawRect(0, 0, 51, 11);
        getImage().setColor(Color.ORANGE);
        getImage().fillRect(1, 1, health, 10);
    }
    /**
     * Η μέθοδος act καλείται κάθε frame και ανανεώνει την εικόνα της μπάρας υγείας για να αντικατοπτρίζει την τρέχουσα υγεία του παίκτη.
     * Επίσης, το αντικείμενο τοποθετείται στη θέση του παίκτη, και η μέθοδος loseHealth ελέγχει εάν ο παίκτης πλήττεται από έναν ζόμπι,
     * μειώνοντας την υγεία κατά ένα.
     */
    public void act()
    {
        setImage(new GreenfootImage(52, 12));
        getImage().drawRect(0, 0, 51, 11);
        getImage().setColor(Color.ORANGE);
        getImage().fillRect(1, 1, health, 10);
        World world = getWorld();
        MyWorld myWorld = (MyWorld) world;
        setLocation(myWorld.getPlayer().getX() - 5, myWorld.getPlayer().getY() - 50);
        loseHealth();
    }
    /**
     * Η μέθοδος loseHealth ελέγχει εάν ο παίκτης πλήττεται από έναν ζόμπι, μειώνοντας την υγεία κατά ένα.
     * Εάν η υγεία πέσει κάτω από το μηδέν, εμφανίζεται το μήνυμα ήττας.
     */
    public void loseHealth()
    {
        World world = getWorld();
        MyWorld myWorld = (MyWorld) world;
        if (myWorld.getPlayer().HitByZombie())
        {
            health--;
        }
        if (health <= 0)
        {
            getWorld().showText("You Lose.. \n You survived for " + (myWorld.mainPlayer.time / 60) + " seconds", getWorld().getWidth() / 2
            //Greenfoot.stop();
        }
    }
}
```

```
getWorld().showText("You Lose.. \n You survived for " + (myWorld.mainPlayer.time / 60) + " seconds",
getWorld().getWidth() / 2, getWorld().getHeight() / 2);
```

Η κλάση YouLose υλοποιεί ένα αντικείμενο που χρησιμοποιείται για την εμφάνιση του μηνύματος "You Lose" στο παιχνίδι όταν ο παίκτης χάνει. Η μέθοδος act της κλάσης είναι άδεια, διότι οι λειτουργίες της συγκεκριμένης κλάσης καλούνται και δημιουργούνται από άλλη ή /και άλλες.

```java
import greenfoot.*;
public class YouLose extends Actor
{
    public void act()
    {
        // Add your action code here.
    }
}
```

Η διαφορά public HealthBar() με public void HealthBar() είναι..

Στη γλώσσα προγραμματισμού Java, η διαφορά μεταξύ της δήλωσης public HealthBar() και public void HealthBar() αφορά τον τύπο της μεθόδου και την τιμή που επιστρέφει.

Η δήλωση public HealthBar() υποδηλώνει τον κατασκευαστή (constructor) της κλάσης HealthBar. Ο κατασκευαστής είναι μια ειδική μέθοδος που ονομάζεται όπως η κλάση και χρησιμοποιείται για την αρχικοποίηση των μελών της κλάσης κατά τη δημιουργία ενός νέου αντικειμένου. Δεν έχει επιστρεφόμενη τιμή (δηλαδή, void), καθώς ο κατασκευαστής δεν επιστρέφει τίποτα.

Αντίθετα, η δήλωση public void HealthBar() υποδηλώνει μια μέθοδο με όνομα HealthBar, η οποία δεν επιστρέφει τίποτα (void). Αυτή η μέθοδος μπορεί να χρησιμοποιηθεί για την εκτέλεση ενεργειών, αλλά δεν είναι κατάλληλη για τη δημιουργία νέων αντικειμένων ή την αρχικοποίηση των μελών μιας κλάσης όπως ο κατασκευαστής.

Συνοπτικά, ο κατασκευαστής είναι μια ειδική μέθοδος που χρησιμοποιείται για τη δημιουργία νέων αντικειμένων, ενώ οι μέθοδοι με τύπο void χρησιμοποιούνται για την εκτέλεση ενεργειών χωρίς επιστροφή τιμής.

Στην προγραμματισμό, ο τύπος void χρησιμοποιείται για να υποδείξει ότι μια συνάρτηση ή μέθοδος δεν επιστρέφει τίποτα. Ο λόγος για αυτό είναι ότι η εκάστοτε συνάρτηση ή μέθοδος εκτελεί μια σειρά από ενέργειες, αλλά δεν παράγει κάποια τιμή που θα χρειαζόταν να επιστραφεί.

Αντίθετα, μια συνάρτηση που έχει έναν τύπο επιστροφής (π.χ., int, double, String κλπ.) σημαίνει ότι αναμένεται να επιστρέψει μια τιμή του συγκεκριμένου τύπου μετά την εκτέλεσή της.

Η χρήση του void σε μια συνάρτηση ή μέθοδο έχει συχνά να κάνει με το γεγονός ότι η συγκεκριμένη συνάρτηση εκτελεί μια εργασία ή δράση, αλλά δεν παράγει μια τιμή που θα χρειαζόταν να επιστραφεί για περαιτέρω χρήση. Για παράδειγμα, μια μέθοδος μπορεί να χρησιμοποιηθεί για να εμφανίσει κάτι στην οθόνη, να αλλάξει καταστάσεις, ή να εκτελέσει άλλες ενέργειες χωρίς να παράγει κάποια συγκεκριμένη επιστροφή τιμής.

Αναφορές

- https://www.youtube.com/@CodingClub
- https://www.youtube.com/watch?v=LT6IPr2W8dw (ανακτήθηκε στις 14/11/2023)