Yordan Neshev
260587938

CRITTER GROUP GENERATOR

In order to create a critter group generator, I had to merge together several classes of components such that the critters themselves, the generator class that creates a group of separate instances of critters, the player whose score is affected by the critters behavior and state as well as the driver class that allowed me to test the functionality and the correctness of the solution.

Let's start by examining the game rules according to which the solution was developed. Firstly, a critter is a unit that is characterized by several attributes including strength, hit points, reward, speed, level, position on the screen as well as living state. As a critter moves along the path at a certain speed, it can get attacked by towers who decrease its hit points. If the critter's hit points reach zero, the critter dies and disappears from the screen awarding the player a certain amount of coins equal to the reward attribute of the dead critter. However, if a critter survives and makes it to the end of the path, the player loses a certain amount of coins from its stash according to the strength of the unit that just exited the path. As the level of the game increases, the critter's attributes scale proportionally to the level which makes it harder for the player. Secondly, a critter group generator is a component that instantiates a number of critters according to the game level and links them together in a list of moving objects. As the game progresses, all objects inside the critter group move according to their speed. Critters are allowed to move only on path cells and do not attack towers around them.

In order to solve the problem stated above, I implemented several classes including Moveable, Critter, CritterGroupGenerator and Player.

Moveable is an abstract class that contains an abstract "move" method and a "collisionWith(other)" method. This class serves as a parent class to all moveable components inside the game. Child components including the class "Critter" extend "Moveable" and are obligated to implement their own move method. This allows to store all Moveable objects inside an array and call their move methods in the game thread to update their position on the screen. The "collistionWith(other)" method is implemented for later when we will use JComponent to represent each object on the screen. This method will allow to detect collisions between different components, for example a missile coming from a defending tower towards a critter. As such, the Moveable class extends JComponent.

Critter is a child class of Moveable. It overrides the move method and implements its own move pattern for the critters. This class contains all the private attributes related to each critter unit as well as all the constructors, getter and setter methods that allow to access them. Several constructors are available in order to allow for customization of the critter's attributes. The default constructor initializes all attributes to their default values and serves as a constructor for a

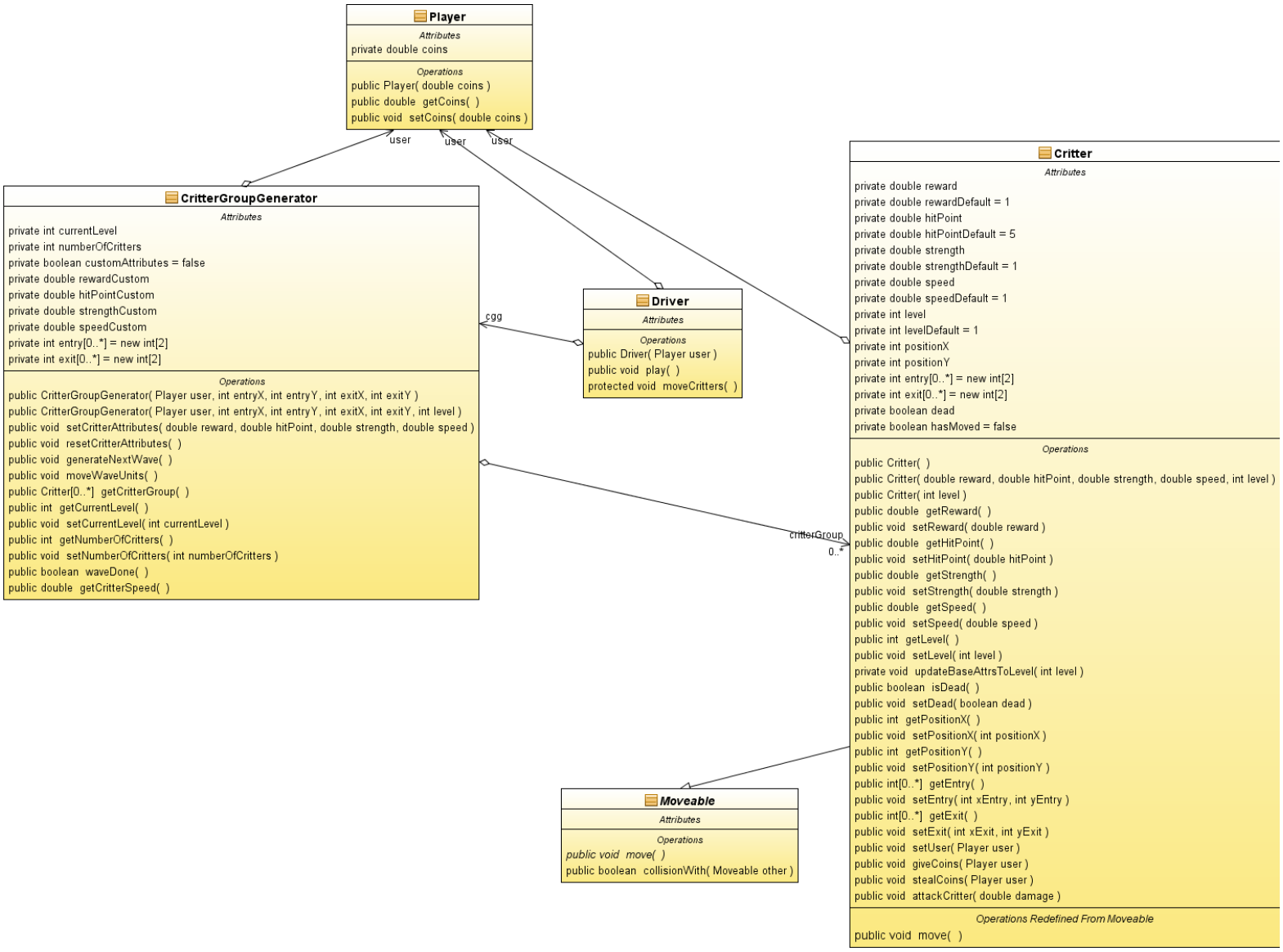
basic first level critter. The level constructor allows to create a critter of a certain level and scale afterwards all of its attributes according to the level passed as a parameter. The third constructor is the one that allows for most customization as we can enter custom values for all the attributes of the critter we are about to create. In the final game, we will most probably make use only of the level scaling constructor in order to reduce the player's influence on the performance and strength of the critters he plays with. The move method of a critter is currently developed in a manner that considers all tiles as path cells. This is fine for the current state of the project considering that there is no defined map yet. The way the move method performs is that it compares the current x and y coordinates of the critter with the x and y coordinates of the exit point and moves the critter horizontally or vertically by one unit until it reaches the exit point. This method should be modified once a map is available. In the map class, we will have a method that simply tells the critter which is the next cell he is supposed to go to. If the critter exits the map alive, the player loses points according to the strength of the critter. I have also implemented giveCoins and attackCritter methods that allow to give coins to the user if he kills a critter and to attack a critter by a certain amount of damage that depends on the tower's strength.

CritterGroupGenerator is a class that allows to generate a wave of critters according to the current level of the game and the desired critter's attributes. As well as before, the default level constructor of the class will mainly be used as we do not want the player to have too much influence on the critter's attributes. An additional method for critter attributes customization was created simply for testing purposes. The main methods of this class are "generateNextWave" and "moveWaveUnits". The first one allows to create a list of a predetermined number of critters with corresponding attributes taking into account the current level of the game. The second method allows to call the move method of all units that are currently instantiated and present in the critter group.

Player is a simple class that allows to store and modify the number of coins the player has at every moment during the game.

In order to test all those individual components, the Driver class implements simple testing routines that generate two waves of critters, one with custom attributes and one with default ones while testing the attackCritter, giveCoins and stealCoins methods on both. The moving routine is tested through custom entry and exit cell coordinates as well as a thread that goes through every step of the critter's movement in the map until it reaches the exit. These are the main components that needed to be tested. When a map as well as tower classes are developed, we will be able to test the global behavior of the program.

The next page contains a UML Class Diagram that illustrates the relation between the classes stated above. A Javadoc documentation has also been generated and provided.


Player
Attributes
private double coins
Operations
public Player(double coins)
public double getCoins( )
public void setCoins(double coins)
user
user
user
CritterGroupGenerator
Attributes
private int currentLevel
private int numberOfCritters
private boolean customAttributes = false
private double rewardCustom
private double hitPointCustom
private double strengthCustom
private double speedCustom
private int entry[0..*] = new int[2]
private int exit[0..*] = new int[2]
Operations
public CritterGroupGenerator(Player user, int entryX, int entryY, int exitX, int exitY)
public CritterGroupGenerator(Player user, int entryX, int entryY, int exitX, int exitY, int level)
public void setCritterAttributes(double reward, double hitPoint, double strength, double speed)
public void resetCritterAttributes( )
public void generateNextWave( )
public void moveWaveUnits( )
public Critter[0..*] getCritterGroup( )
public int getCurrentLevel( )
public void setCurrentLevel(int currentLevel)
public int getNumberOfCritters( )
public void setNumberOfCritters(int numberOfCritters)
public boolean waveDone( )
public double getCritterSpeed( )
cgg
Driver
Attributes
Operations
public Driver(Player user)
public void play( )
protected void moveCritters( )
critterGroup
0..*
Critter
Attributes
private double reward
private double rewardDefault = 1
private double hitPoint
private double hitPointDefault = 5
private double strength
private double strengthDefault = 1
private double speed
private double speedDefault = 1
private int level
private int levelDefault = 1
private int positionX
private int positionY
private int entry[0..*] = new int[2]
private int exit[0..*] = new int[2]
private boolean dead
private boolean hasMoved = false
Operations
public Critter( )
public Critter(double reward, double hitPoint, double strength, double speed, int level)
public Critter(int level)
public double getReward( )
public void setReward(double reward)
public double getHitPoint( )
public void setHitPoint(double hitPoint)
public double getStrength( )
public void setStrength(double strength)
public double getSpeed( )
public void setSpeed(double speed)
public int getLevel( )
public void setLevel(int level)
private void updateBaseAttrsToLevel(int level)
public boolean isDead( )
public void setDead(boolean dead)
public int getPositionX( )
public void setPositionX(int positionX)
public int getPositionY( )
public void setPositionY(int positionY)
public int[0..*] getEntry( )
public void setEntry(int xEntry, int yEntry)
public int[0..*] getExit( )
public void setExit(int xExit, int yExit)
public void setUser(Player user)
public void giveCoins(Player user)
public void stealCoins(Player user)
public void attackCritter(double damage)
Operations Redefined From Moveable
public void move( )
Moveable
Attributes
Operations
public void move( )
public boolean collisionWith(Moveable other)