Revised Program Specifications

Space Adventure

Team: Cindy (Xinzhu) Fang, Jenny Kim, Justin Warring
May 2, 2018

Project Description

Our program will code for a game that involves the player controlling a spaceship in space. Upon running the program, the player will be able to bring up a window that allows them to play the game.

This will bring up an instructions window that will tell the user what to do. The user will be able to control the player ship using the movement keys: w,a,s,d and the j key to shoot at incoming asteroids. They can also rotate the ship in either the clockwise or the counter clockwise direction by holding down the q or e keys.

They will also be able to rescue other spaceships by touching the spaceships on the planets. The spaceship will then move to behind the player's ship. Each subsequent rescue will result in the addition of another ship, eventually creating a long chain of ships that must be protected. Rescuing a ship will reward the player with 1 life but allowing them to get hit by an asteroid will result in the loss of a point, thus some considerations must be made to on whether to rescue or not.

The player will initially start with 5 lives, which will be updated on the screen and when all the lives are gone, the game will end. The game will slowly increase in difficulty as time passes by increasing the frequency of asteroids and thus the amount of distracting debris in the area as well.

This program will serve as a new and interesting game that incorporates various elements of old school Atari games for people to enjoy in their free time.

Diagram

There are several classes of different objects that the player can interact with. There is the Ship class, which the player ship and the other trailing ships are made from, the asteroid class, and the planet class (debris and bullet are extended from asteroid). These three objects are subclasses of a class called Sphere, which gives each object its physical properties in the game (position, acceleration, radius, velocity, etc). Each object has a method that allows the updating and the drawing of itself. Inside the world class, there are methods that take the linked list that each object is in (excluding Ship and Planet since there are only single instances of those) and draws them.We also check collisions in World between asteroid and bullet, asteroid and player ship, asteroid and trailing ship, and asteroid and asteroid. If they do collide, we then remove the objects from our linked list. The main class contains classes that implement Runnable, updatingSpheres and checking collisions. This is also where the difficulty slowly is increased to allow for more frequent and faster asteroid/planet generation and movement. KeyListener is used here to help control the player ship and the world is also repainted. The paintComponent method is used to display the amount of lives that a player has left and forces the System.exit(0) when there aren't any more left.

Screenshots:

Initial Starting Screen/Instruction Message

Trailing ships

Exit Message

Classes

- Class
  - Data Structure
    - Pair: many instances
    - Linked List: many instances
  - Entities
    - Sphere: many instances
    - Asteroid: many instances
    - Bullet: many instances
    - Debris: many instances
    - Ship: few instances
    - MyShip: One instance
    - Planet: One or zero instance
  - high-level
    - World: contain all the entities
    - Space: running multiple threads, which call different methods in World at their own pace. One instance
  - Threads in Space
    - Runner: do everything except what the other two do. One instance
    - KeepPlanetComing: One instance
    - KeepAsteroidComing: One instance

	Constructor	fields	methods

Pair	public Pair(double initX, double initY){
	x = initX;
	y = initY;
	}
		public double x;
		public double y;
			add()
			times()

Linked list	public CindyDS()	public Node<E> end;	append() length() remove() get()
Sphere	public Sphere(int initWidth, int initHeight,int	static int height; static int width; static int margin;	public void update(World w, double time)

	initMargin, int initDiff)
	public Sphere()
		static int diff;
		Pair position;
		Pair velocity;
		Pair acceleration;
		double radius;
		Shape myShape;
		Random rand
			public void drawShape(Graphics 2D g2D)
			
			public void updateShape(){
			}
			
			public boolean checkDeath()

Asteroid extends Sphere	public Asteroid(){	int NumAxis
		int NumVertices
		double speed;
		double angle;
		double radius
		double[] angles
		double[] dists
		int[] xs
		int[] ys
			public void update(World w, double time)
			
			public void drawShape(Graphics 2D g2D)
			
			public void updateShape(){
			}

Bullet extends Asteroid	public Bullet(double x, double y, double a){	double radius	public void update(World w, double time)
			
			public void drawShape(Graphics 2D g2D)
			
			public void updateShape(){
			}

Debri extends Asteroid	public Debris(double x, double y){		
Ship extends Sphere	public Ship (int initWidth, int	Pair acceleration double angle	public void update(World w,

	initHeight,int initMargin, int initDiff){
	super(initWidth, initHeight, initMargin, initDiff);
	
	}
	
	public Ship(Pair planetPosition)
		double rotation
		double radius
		int[] xs;
		int[] ys;
			double time)
			
			public void drawShape(Graphics 2D g2D)
			
			public void updateShape(){
			}

MyShip extends Ship	public MyShip (int initWidth, int initHeight,int initMargin, int initDiff){		public void update(World w, double time)
			
			public void updateShape(){
			}
			
			public void roundScreen(){

Planet extends Sphere	public Planet(){	double radius	public void update(World w, double time)
			
			public void drawShape(Graphics 2D g2D)
			
			public void updateShape(){
			}

World	public World(int initWidth, int initHeight,int initMargin, int initDiff){	int height;
		int width;
		int margin;
		int diff;
		int lifeCount
		
		int numAsteroids;
		int numBullets;
		int numSpheres;
			//draw myShape for all entities
			public void drawSpheres(Graphics2D g2D){
			
			// helper method for drawSpheres(), draw entities stored in linked list

		int numShips;
		CindyDS<Sphere> asteroids =
		Planet planet;
		MyShip myShip;
		
		Boolean removingAst
		CindyDS<Sphere> allDebris=
		CindyDS<Sphere> bullets =
		CindyDS<Sphere> freeShips =
		CindyDS<Sphere> capturedShips =
			public void drawList(CindyDS list, Graphics2D g2D){
			
			// helper method for updateSphere(), update entities stored in linked list
			public void updateList(CindyDS list,double time){
			
			// helper method for updateSphere(), draw captured ships
			public void updateCapturedShips (double time){
			
			// update all entities
			public void updateSpheres(doubl e time){
			
			// create a new planet, to be called by KeepPlanetComing thread
			public void renewPlanet(){
			
			// create a new asteroid, to be called by KeepAsteroidsComin g
			public void addAsteroid(){
			
			// capture freeShip when myShip

			overlaps with it
			  public void capture(){
			
			// check collision among bullets, debris, asteroids and ships
			  public void checkCollision(){
			
			  // helper method for checkCollision
			  public int[] magicSort(ArrayList <Integer> temAL){

Space	public CindySpace(){	public static final int WIDTH
		
		public static final int HEIGHT
		
		public static final int MARGIN
		
		public static final int FPS
		
		public int diff
		
		public static boolean paused
		
		double diffThreshold
		
		Int i
		
		Boolean key1/2/3/4
			public static void main(String[] args){
			
			public void paintComponent(Graphics g) {
			
			public void keyPressed(KeyEvent e) {
			
			public void keyReleased(KeyEvent e) {
			
			public void keyTyped(KeyEvent e) {

		World world Random rand	
class KeepAsteroidsComing implements Runnable{			public void run()
class KeepPlanetComing implements Runnable{			public void run()
class Runner implements Runnable {			public void run() public void incDiff(){

Entities have fields according to their body and behaviors. For example, world has fields of entities and space has fields about the game. All the fields and methods are public, because this is just a game, there is no safety issue.

Most classes have one constructor. Ship has two, one for myShip, one for other ships. Sphere has two, one for the ship constructor for myShip constructor, one for other entities. There is this special channel for the four parameters: height, width, margin, and diff to be passed from Game to World to MyShip (the most important one) then to the static fields of Sphere to be shared among all entities, by first calling the special Ship constructor and then calling the special Sphere constructor.

The four methods update(), updateShape(). drawShape(), and checkDeath() in Sphere, are the only methods World is calling from outside. The first three are needed for all children and grandchildren of Sphere, which are overridden in some entities. The last one is needed by

everyone except myShip and captured ships. Some entities have helper methods for these four methods.

Everything in the world is actually the subclass or subsubclass of Spheres. Nonetheless, all the methods are there in Sphere even if they are empty. This way, in World, a call on Sphere can be applied to any entity. Accordingly, all entities are cast to Sphere in the fields of World. Six methods of World: updateSphere(), updateKey(), shoot(), capture(), checkCollision(), and drawSpheres() are called by Runner thread in Space; addAsteroid() is called by KeepAsteroidsComing thread; renewPlanet() is called by KeepPlanetComing thread. World has some helper methods for the above methods: drawList() is called by drawSpheres(); updateList() and updateCaptureedShips() are called by updateList(); magicSort() is called by checkCollision(). Among these, updateSphere(), updateList(), updateCaptureedShips() capture(), checkCollision(), drawSpheres(), and drawList() are recursive.