Senior Design Fall 2019
Team 4
Client-Server Documentation

Luis Bueno, Vagan Grigoryan, Tobias He, Andy Wang

Player.java

- public class Player implements Serializable
    - This class represents each user that connects. It implements Serializable, which in Java allows instances of this class to be sent and received with Object Streams.
        - public Player(double x, double y)
            - A simple constructor that spawns a new Player object at the coordinates provided. It also initializes empty vectors that store the users' thrown projectiles and collected loot.
        - public static final int WINDOW_WIDTH
        - public static final int WINDOW_HEIGHT
            - These values set the size of the window created by the Window class. They are used in many computations in the Player class. They were localized to the Player class instead of the Window class so that if the GameServer is on a server cloud (like Amazon AWS EC2), the Window class does not need to be on that server cloud.
        - private boolean isActive
            - This boolean was added so that when a player disconnects, they stop being rendered. This value is linked to rendering in the Window class.
        - private int ID
            - Each Player receives a unique ID. This is used for consistency in terms of updating the central game state.
        - protected double x
        - protected double y
        - protected double dx
        - protected double dy

        - These variables are used in the manipulation of the Player's position. The variables dx and dy are rates of change.
- private String graphic
    - This String represents the file name that will be displayed when this Player is rendered.
- private Vector<Projectile> projectiles
- private Vector<Projectile> inventory
    - For convenience in terms of Object Streaming, the projectiles a Player has thrown and picked up are contained within each Player.
- public Vector<Projectile> getProjectiles()
    - This method returns the projectiles vector of a Player, which otherwise has private access.
- public void setMovement(double dx, double dy)
    - This method sets the rates of change for this Player's x and y positions.
- public double getX()
- public double getY()
- public double getDX()
- public double getDY()
- public String getGraphic()
- public int getID()
    - These methods are all simple "getters" that return the values of the Player's private access variables.
- public void setGraphic(String graphic)
    - This method sets the private access variable graphic.
- public void setID(int ID)
    - This method sets the private access variable ID.
- public void setActive()
- public void setInactive()
- public boolean isActive()

            - These methods set and get the value of the private access variable isActive, which is used to ensure disconnected players immediately stop being rendered.
        - public void move()
            - This method performs various tests that ensure the Player has not moved out of bounds. Then, if it's appropriate, it changes the x and y position of the Player by the dx and dy values. Additionally, it moves all of the Player's thrown projectiles; if a Projectile has reached its max range and should disappear, this is where that is checked and the Projectile is removed from the Player's Projectiles vector if that is the case.

Projectile.java

- public class Projectile implements Serializable
    - This class represents Projectiles thrown as well as Projectiles that are able to be picked up as loot (and then thrown). It implements Serializable, which in Java allows instances of this class to be sent and received with Object Streams.
        - public Projectile(double x, double y, String graphic)
            - This constructor is for static Projectiles (loot). It automatically sets the "lootable" variable to true.
        - public Projectile(int sourceID, double x, double y, double dx, double dy, String graphic)
            - This constructor is for thrown Projectiles. As a result, it has parameters for setting rates of change, as well as a sourceID that stores which Player threw this mobile Projectile.
        - public static double MAX_FLY

- public static double FLY_SPEED
    - These values represent the maximum distance this Projectile will move before being deleted, and the speed at which this Projectile moves.
- private int sourceID
    - This variable stores the ID of the Player that threw this Projectile.
- protected double x
- protected double y
- protected double dx
- protected double dy
    - These variables are used in the manipulation of the Projectile's position. The variables dx and dy are rates of change.
- private String graphic
    - This String represents the file name that will be displayed when this Projectile is rendered.
- private boolean lootable
    - This variable represents if this Projectile is one that was thrown or one that was spawned by the Server for Players to pick up as loot.
- public void setStats()
    - Based on the graphic variable of this Projectile, this method will set the maximum range it can be thrown before disappearing, and the speed at which it moves.
- public boolean move()
    - Firstly, this method determines if this Projectile has reached its maximum range and should be deleted. In this case it does nothing but return false. If this is not the case, it continues the Projectile's flight, then returns true.
- public double getX()

        - public double getY()
        - public String getGraphic()
            - These methods are simple “getters” that return the values of the private access variables.
        - public void setGraphic(String graphic)
            - This method sets the private access variable graphic.
        - public void setMovement(double dx, double dy)
            - This method sets the rates of change for this Projectile’s x and y positions.

GameServer.java

- public class GameServer
    - This class runs a Server that accepts incoming connections and handles game state.
        - static ExecutorService pool = Executors.newFixedThreadPool(32)
            - This is a thread pool with a limit of 32. The limit can be changed, we decided on 32 as a fine amount of Players considering the map size.
        - protected static PlayerTable table = new PlayerTable()
            - This is the central Player table. All Projectiles are contained within each Player object, so essentially this contains the entire game state excluding spawned loot.
        - static int ID = 1
            - This variable is used to set Players’ ID values after they join.
        - static Vector<Integer> disconnectedIDs = new Vector<Integer>()

            - This vector allows for ID recycling. In other words, Players that disconnect have their IDs added to this vector so that other Players that connect can reclaim the ID values.
        - static Random random = new Random()
            - This allows for randomization in the spawning of loot.
        - public static class LootSpawn implements Runnable
            - This method attempts to spawn a new loot at a random location, as long as there aren't too many loots available already on the map.
        - public static void main(String[] args) throws Exception
            - This method opens a ServerSocket that allows new Players to join, and spawns a thread for each Player that connects.
        - private static synchronized int generateID()
            - This method generates new IDs for incoming Players, or assigns them a "recycled" ID left behind by a disconnected Player. It is synchronized, which in Java means only one thread can run this method at one time. This prevents consistency issues in terms of assigning new IDs.
- private static class Connection implements Runnable
    - This is what each thread runs when it is spawned.
        - Socket socket
        - ObjectInputStream in
        - ObjectOutputStream out
        - int ID
            - These variables allow for two-way communication with each Player, and keep track of their ID.
        - long timeClientConnected = System.currentTimeMillis()
            - This variable stores the time when this Player connected.

        - long timeThreadWasBusy = 0
            - This variable is initialized to 0 and is added to each time the Server is performing work to send and receive information from each Client.
        - public Connection(Socket socket) throws IOException
            - This constructor sets this Client's Socket and generates an appropriate ID for them. It also initializes the Object Streams based on the Socket's connection information.
        - public void run()
            - This method contains the main work that the GameServer does. It keeps track of time spent communicating with the Client, and reads in the Player's local game state, and then sends the Player the global game state.
            - After disconnecting, statistics are printed about that Clients' connection and usage. Then the Socket is closed and the ID is set up to later be recycled (assigned to a new Player).

PlayerTable.java

- public class PlayerTable implements Serializable
    - This class stores a vector of Player objects. It performs some useful special functions like updating a Player's state and adding new Players in the right location. It implements Serializable, which in Java allows instances of this class to be sent and received with Object Streams.
        - private Vector<Player> table
            - This is the main table inside this class that stores all the Player objects. Note that all Projectiles are stored within each Player object.

        - private Vector<Projectile> loot
            - For convenience in terms of Object Streaming, the loot on the map is contained in the PlayerTable class.
        - public PlayerTable()
            - This method simply initializes the Player and Projectile vectors to empty vectors.
        - public Vector<Player> getTable()
        - public Vector<Projectile> getLoot()
            - These methods are simple “getters” that return the value of the private access vectors mentioned above.
        - public int getSize()
            - This method returns the size of the vector of Players.
        - public void update(Player player)
            - This method performs some logic. It checks if the Player’s ID is larger than the current size of the Player vector. If it is, it appends the Player parameter. If it is not, it sets the appropriate index in the vector to this Player based on ID.

Window.java

- public class Window extends JFrame
    - This class performs window spawning and graphics operations such as drawing images and clearing images.
        - private GameCanvas canvas
            - The main canvas in this JFrame.
        - PlayerTable table
            - The PlayerTable to render. Note that loot that has spawned on the map is contained within a PlayerTable, and that Projectiles thrown by Players are contained within those Players.

        - public Window()
            - Initializes a new JFrame with the GameCanvas as its primary element. It sets the default operation, which is EXIT_ON_CLOSE (in Java meaning completely exit the program once the 'X' on the window is pressed). Also it sets the title of the window, and configures the window so that resizing is not allowed. This is so all Players have the same view of the map.
        - public GameCanvas getCanvas()
            - This method returns the value of the private access variable canvas mentioned above.
        - public void setTable(PlayerTable table)
            - This method sets the PlayerTable to be rendered, and then immediately calls repaint(), a method that redraws all the elements in the (now updated) PlayerTable.
- private class GameCanvas extends JPanel
    - This class has only one method, which is an @Override. This method is paintComponent, which runs when repaint() is called. In this method, we render all objects that are active (recall the Player object has a variable isActive). In the case that a Player is no longer active based on the variable isActive, that Player's character and Projectiles thrown are not rendered. All of the loot that has spawned on the map is also rendered in this method.

GameClient.java

- public class GameClient
    - This class handles all of the heavy lifting on the Client side. It tracks keyboard input, spawns a new Window, and performs logic to make sure each Client is keeping up to date with the game state.

- private static ObjectOutputStream out
- private static ObjectInputStream in
    - These Object Streams are used to communicate back and forth with the GameServer.
- private static Random random = new Random()
    - Used to select a random position that a new Player spawns at.
- public static Player player = new Player(random.nextInt(Player.WINDOW_WIDTH - 32) + 32, random.nextInt(Player.WINDOW_HEIGHT - 32) + 32)
    - This is the Player that this GameClient belongs to. It is spawned at a random location within the borders of the map.
- public static void main(String[] args) throws Exception
    - This method does the heavy lifting in the GameClient class. It opens a new Window, a new Socket, and sets all the Object Streams. Then it receives the ID the server has assigned, and begins listening for mouse and key events. Then a new scheduled task is run, which is described below.
- Runnable task
    - This task does all of the two-way communication. It sends the GameServer the local game state, and receives the global game state. After each iteration of two way communication, it calls the Window setTable(PlayerTable table) method, which updates the PlayerTable in the Window class. Note that when this happens, repaint() is called, which will redraw the elements in the game.
- private static GameKeyListener keyListener = new GameKeyListener()
- private static GameMouseListener mouseListener = new GameMouseListener()

                - These are instances of customized classes we implemented to track keyboard and mouse input.
- private static class GameKeyListener implements KeyListener
    - This class listens for key pressed and key released events.
        - public void keyPressed(KeyEvent e)
            - This method sets the Player's movement (dx and dy values, the rates of change of x and y) based on which key was pressed.
        - public void keyReleased(KeyEvent e)
            - This method resets the Player to a standing state (dx and dy are 0) when a key is released.
- private static class GameMouseListener implements MouseListener
    - This class listeners for mousedown events.
        - public void mousePressed(MouseEvent e)
            - This method performs vector normalization to determine which direction a newly chosen Projectile should be launched at. A random Projectile is selected each time and added to this Players' Projectiles vector.