Boston University Adventure

Issue & Solution Documentation [Opal]

**Client**

**Graphics**

Issue: How is the map rendered? [C.G.000]

Solution: The map is rendered by the GameMap’s render function [C.G.001] & the UIEngine’s render function. [C.G.002]

Issue: How often is the map rendered/what triggers a render? [C.G.010]

Solution: The map is rendered according to CoreGameLogic’s renderLoop function [C.G.011], which is called on each iteration of the infinite loop in Main. [C.G.012]

Issue: How does the client know where to render characters on the map? [C.G.020]

Solution: Characters, known as “Actors” are represented by animated sprites (AnimatedSprite.java). ActorEngine.java stores a list of all actors, and Actor.java stores data for the actor and contains its render function. In other words, each character is rendered individually on top of the map [C.G.021].

**Networking**

Issue: How does the client join a game? [C.N.000]

Solution: A client joins a game by entering their name as well as the server’s IP address at a screen generated by LoginScreen.java. Once this information is submitted [C.N.001], a NetworkEngine is created and it uses a NetworkStreamWriter to send a login request. [C.N.002].

Issue: How does the client receive the map? [C.N.010]

Solution: The client receives the map via the network in packets with an opcode of MI for map image and MD for map data. These are processed in NetworkEngine [C.N.011], and then depending on the opcode, the appropriate retrieve function is called from the NetworkStreamParser, getMapData() [C.N.012] or getMapImage() [C.N.013].

Issue: How does the client know when the game starts? [C.N.030]

Solution: The client receives countdown packets from the server and processes them with NetworkEngine [C.N.031]. The server won’t permit moving until the game has begun, so the client doesn’t have to block until the game starts.

Issue: How does the client tell the server it wants to move? [C.N.020]

Solution: As the client’s coreGameLogic’s renderLoop runs, it checks keyboard input and processes it with processInput [C.N.021], which in turn calls the NetworkEngine’s NetworkStreamWriter’s sendActorMove. [C.N.022]

Issue: How does the client receive information about other characters? [C.N.040]

Solution: NetworkEngine processes new actor data packets (opcode NA) [C.N.041] using NetworkStreamParser’s getNewActorData() which returns an actor. This actor is then added to the game’s instance of ActorEngine.

**User Interface**

Issue: How does the client setup the user interface? [C.U.000]

Solution: The user interface is setup first by the CoreGameLogic class based upon the state of the game [C.U.001], resulting in handing UI setup to one of three classes:

LoginScreen.java (LOGIN\_STATE) creates a background [C.U.001], input fields with labels [C.U.002] and a login button [C.U.003], and waits for it to be pushed to attempt a connection. [C.U.004]

LoadingScreen.java (LOADING\_STATE) displays a black background [C.U.005] and renders a progress bar atop [C.U.006] it as the connection is established and the map data downloads. The bar gains progress when the NetworkEngine receives appropriate packets for the map. [C.U.007], [C.U.008].

UIEngine.java (INGAME\_STATE) renders the data bar with player image, name, message and credits [C.G.002], all necessary information for the user.

Issue: How are key press events detected and processed?

Solution: See solution to issue [C.N.020].

**Server**

**Networking**

Issue: How do clients join a game? [S.N.000]

Solution: Clients join a game by requesting a connection to the server and then sending a “login request” packet. Main recognizes this incoming connection and establishes a socket [S.N.001] and then establishes a ClientHandler thread for each client. [S.N.002]

Issue: How is the map transmitted? [S.N.010]

Solution: After the server receives a login request from a client [S.N.011] it uses a NetworkStreamWriter to send both the map image [S.N.012] and data [S.N.013] files.

Issue: How and when are actor positions transmitted? [S.N.020]

Solution: Actor positions are initially transmitted when the countdown ends by ClientHandler (which is blocking after processing the login packet until the game begins) [S.N.021]. After that, GlobalGameLogic transmits all moves – as it iterates through all actors, if an actor moves it transmits that move to all players. [S.N.022] (NPCs), [S.N.023] (PlayerCharacters)

**Game Logic**

Issue: How do non player characters move? [S.L.000]

Solution: Non player characters may either move vertically or horizontally at any given moment, but not both at the same time. In fact, NPCs alternate between moving vertically and horizontally. A non player character’s position is changed the same way a player character’s is – GlobalGameLogic iterates through all movable characters and moves actors toward their “move to” position based upon speed and time elapsed. If a NPC who should be moving has the same move to position as their current position, NPCEngine’s generateNewPosition(Integer id) is called [S.L.001]. This function calls GameMap’s getRandomVertical or getRandomHorizontal function, depending on what direction it should move next, stored in NonPlayerCharacter’s nextDirection. [S.L.002]

Issue: How are the contents of a manhole decided? [S.L.010]

Solution: The contents of a manhole are decided based on a probability calculation performed each time there is an interaction with a manhole in PlayerCharacter’s processCollision(). [S.L.011]

Issue: How are collisions (interactions) between objects detected? [S.L.020]

Solution: The run() function within GlobalGameLogic contains an infinite loop [S.L.021]; each time through this loop, every movable actor has its position updated. After that, any player character who moved checks to see if it is in the same cell as another actor.

Issue: How are collisions processed?

Solution: The processCollision(Integer ActorID) function in Actor processes what happens when the actor collides with the actor who’s id is in the argument. Depending on that actor’s type, this actor’s health and credits change accordingly.

Issue: How does the server start a game? [S.L.030]

Solution: The server starts a game by changing the game state to INGAME. This occurs when the countdown in GlobalGameLogic reaches 0. [S.L.031]

Issue: How does the server end a game? [S.L.040]

Solution: The server ends a game in GlobalGameLogic by breaking out of the game loop [S.L.041], transmitting a “gameover” packet to all clients and then closing their socket connections. [S.L.042]

Issue: How are player starting locations determined? [S.L.050]

Solution: Player starting locations are picked at random using the getRandomMapPoint() function in GameMap. This happens when the ClientHandler receives a login packet (opcode LG). [S.L.051]

Issue: How does a player move? [S.L.060]

Solution: Each actor has a current position (position) and a target position (moveto) [S.L.061]. GlobalGameLogic iterates through all players and moves them toward their target position based upon their speed and how much time has elapsed since they last were moved [S.L.062]. Moveto is updated by the ClientHandler [S.L.063].

**Data Storage**

Issue: How are players represented?

Solution: Players are represented by the PlayerCharacter class, which extends Actor. The ActorEngine maintains a list of all Actors.

Issue: How are professors/viruses represented? [S.D.010]

Solution: Professors and viruses are represented by the NonPlayerCharacter class, which extends Actor. The GlobalGameDatabase maintains a list of all actors. These are imported when GlobalGameLogic loads by NPCEngine’s getNPCsFromFile. [S.D.011]

Issue: How are eating establishments represented?

Solution: See solution to issue [S.D.010]

Issue: How does a NonPlayerCharacter know which direction to move next? [S.D.020]

Solution: NonPlayerCharacter stores the next direction in an enum RELATIVE\_DIRECTION which has two entries VERTICAL(-1) and HORIZONTAL(1). (-1 and 1 are used so that multiplying by -1 will easily switch between them). This is set by NPCEngine’s generateRandomCharacters [S.D.021] and is modified by generateNewPosition [S.D.022].