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)

Issue: How and when are player credits transmitted?

Solution: Same as Actor positions - See issue [S.N.020].

Issue: How does the server wait for clients to connect? [S.N.030]

Solution: The server blocks in Main waiting for 20 seconds [S.N.031] for a client to connect [S.N.001]. By nature of a timeout, the 20 seconds is reset each time a client joins. This is intentional, and users are not permitted to join in the last 10 seconds of a game countdown.

**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]

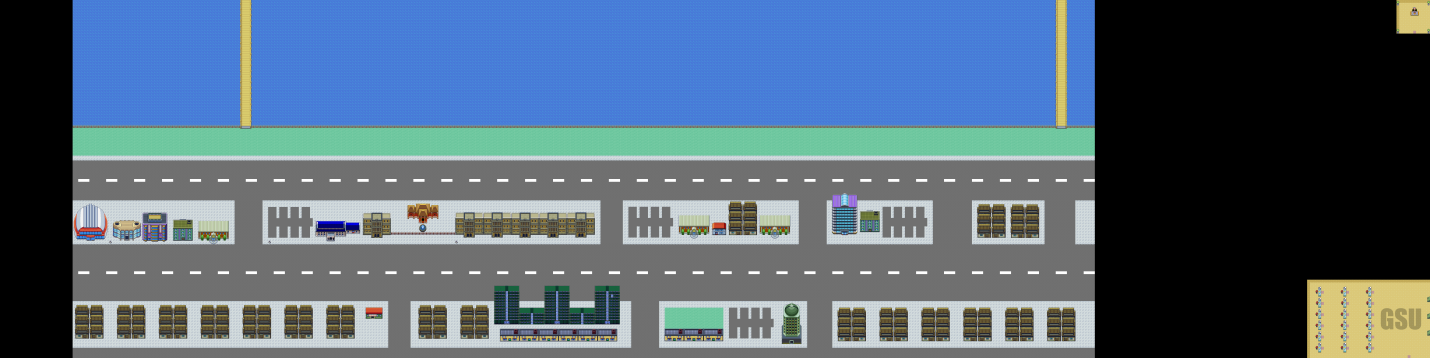
Solution: Player starting locations are picked at random using the getRandomMapPoint() function in GameMap. This happens when the ClientHandler receives a login packet (opcode LG). [S.L.051]

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].

Issue: What is the purpose of “teleport” squares?

Solution: Teleport squares allow a player to be transferred to another section of the map. The idea behind this is that then players can seem to enter buildings. Consider a map like follows:



The areas on the right cannot be walked to nor seen from the main map area, but by teleporting to the GSU area when the player steps on the door to the GSU, it simulates entering. (The player can also step on the door ‘inside’ the GSU to return).

Issue: How does a player increase health? [S.L.070]

Solution: A player increases health by exercising or eating (at a healthy location). This happens when a player enters the same cell that represents a food establishment and the collision is processed. The player must have traveled 200 cells since the last health increase, stored in a PlayerCharacter float datamember [S.L.071]. Depending on the restaurant or exercise mechanism, health is increased accordingly. [S.L.072]

Issue: How does a player attend class and gain course credit? [S.L.080]

Solution: A player attends class by colliding with a cell representing a classroom while not being sick. In order to gain course credit, a class must be attended twice. Classes cannot be attended twice in a row (another class must be attended inbetween). As such, PlayerCharacter contains a datamember to store the last class a player attended [S.L.081], as well as an array counting how many times a player has attended class [S.L.082]. When a player collides with a classroom the player must not be ‘sick’ or else it does not count as attending class [S.L.083]. If not then a check is made to ensure this was not the last class attended [S.L.084], and then attendance is awarded [S.L.085]. If this is the second time attending, the player receives credit [S.L.087].

Issue: How are bridge circuits managed? [S.L.090]

Solution: A map contains only one bridge circuit, and this can only be entered from one cell at each end. These cells are given a type all their own [S.L.091]. When a player collides with one of these cells, the player could be entering or exit the bridge circuit, and if leaving it could be the same way the player came in or a different way. To account for this, each player has a boolean value of whether or not the player is in the bridge circuit [S.L.092]. When a player enters the bridge circuit, the start location is stored and the boolean is set [S.L.093]. When a player exits the circuit, the exit point is compared with the start point [S.L.094] and if they’re different, health is awarded [S.L.095]. Either way, the bridge circuit boolean is reset [S.L.096].

Issue: How is a player’s speed determined? [S.L.100]

Solution: A player’s speed is based upon the player’s health. PlayerCharacter’s updateSpeed() modifies the speed based upon health. [S.L.101] A few exceptions are when a player is sick their speed is constantly slow, and when a player is in a bridge circuit their speed is constantly fast.

Issue: How is the countdown handled? [S.L.110]

Solution: Once an instance of GlobalGameLogic begins running, it loads/generates the NonPlayerCharacters [S.L.111] and then begins a 30-second countdown. This is done by storing the current time [S.L.112] and then waiting until 30 seconds have elapsed [S.L.113]. This way, when a new client joins the countdown can easily be reset just by resetting the starting point. [S.L.114]

**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].

Issue: Actors are identified by a unique id. How is this id generated so that no two actors have the same id? [S.D.030]

Solution: The GlobalGameDatabase provides a mechanism for generating a unique id. It contains an integer member [S.D.031] with the next available id. When an id is assigned, the integer member is incremented [S.D.032].

Issue: How are map cells represented and what are their types? [S.D.040]

Solution: The map itself is represented by two image files. The first is a graphical map that the user will see and interact with. The second image is the same size but only has 3 colors: black, white, red. Squares that an actor cannot walk are black. Normal squares to walk on are white, and things built into the map are red. (e.g. the door to FitRec or a restaurant, or the entrance to a bridge circuit). Based upon these colors, the cells have two types: walkable (1 – [S.D.041]) and not walkable (0 – [S.D.042]), and this is determined by GameMap’s getCellType function. The image for the map is stored in a Bufferedmage [S.D.043]. A cell is 16x16 pixels.