Mohawk College COMP-10246 Capstone Proposal

Project Scope Document Description

My Capstone Proposal

“Dizzy’s Disease” Unity Game

Jin Zhang

000878821

Mohawk College COMP-10246 Capstone Proposal

Project Scope Document Description

Table of Contents

Mohawk College COMP-10246 Capstone Proposal

Project Scope Document Description

My Idea

For my capstone, I want to create a zombie survival-RPG game with a living, dynamic NPC ecosystem and economy. The game will have a character profile for various stats like level, exp, strength that determine carry weight and ability to wear equipment, or accuracy that determines hit chance, etc., with which the user will be able to allocate stat points to upon leveling. The player makes decisions that cost time and other resources affecting the game state and its various aspects. A marketplace exists with outposts where items are sold, which is implemented like an ecommerce shopping site, with the prices determined by events that affect the game state. For example, if an outpost gets attacked and casualties are high, the backend game logic will make it, so meds are consumed by the NPCs at a higher rate and the price increases, and the outposts manpower is depleted. The player will be able to go on expeditions into the inner city to perform missions and loot runs, with their decisions in the inner city affecting the state of the game, like locations being aggro'd and attracting more zombies, or quieter areas that haven't been touched increasing the likelihood of other bandits occupying the location.

My Scope

- have a mobile first, responsive design targeting mobile and desktop
- protect against SQL injection attacks
- have authentication and authorization systems
- consistently use techniques to ensure data integrity and validation
- perform error handling and logging without exposing internal information to the user
- use relational, SQL based databases, no ORM or other framework that abstracts SQL (Hibernate, Django, etc) allowed.
- secure data storage for player account information and any personal data input

Unity Project Scope Elements

All attributes and numbers are placeholders to be balanced with playtesting best on gameplay feel

If the game is based on an existing game, provide the name of the game and the official rules. The official rules may be provided as a URL or as an external PDF file.

Inspired by Survival Horror games like Resident Evil, Last of Us and DeadFrontier.

A colony/outpost/settlement mechanic like the Last of Us Multiplayer. During multiplayer, the player is able to loot settlement resources, with more resources resulting in greater settlement population.

Hosted database driven persistent memory (i.e. leaderboard, character inventory, etc)

Will have a character inventory for the user and NPCs, as well as a marketplace, created with a persistent database that carries over.

Map/Terrain/Board/Environment that the user interacts with the program, including any special interaction zones or attributes.

A hostile zone where the character can go out and has to scavenge for supplies to provide for their settlement where they will encounter enemy NPCs.

Safe zones are Outposts and other settlement or fortified areas of safety where the marketplaces exist and other utility locations.

Player controlled character primary ability, attributes (e.g. health, strength, etc), and level up ability (jump higher, fly, etc)

The player character has RPG mechanics (like in DeadFrontier) that increase with the **player increasing in level by performing the skill**, which will be **resource restricted**

- to level up a skill in shooting for example, the player will need to have a weapon (which will degrade over time and will have maintenance cost) and bullets that they need to expend, so skills cannot just be brainlessly spammed to level up
- experience gains can vary depending on context, (e.g. state of the players nutrition and health, level differences between player and enemy) and with more skillful performances (landing consistent headshots with shorter time between the shots)

Will have weapon proficiency stats:

- Melee skills: Knives, Swords, Blunts
- Firearm Skills: Handguns, Shotguns, Machine Gun, Rifles, Explosives

Survivability Stats

- Nourishment Levels: Will apply debuffs or buffs based on nourishment
- Sleep Levels: Will apply buffs or debuffs to EXP gain based on sleep levels

General Attribute Stats

- **Endurance:** Allows for more actions before depleting energy bar, lower energy bar levels result in fatigue debuffs which can be mitigated with higher endurance
- **Strength:** Allows for greater carrying capacity, less fatigue from carrying weight or swinging heavier weapons, doing more damage with heavier melee weapons like blunt weapons, better handling of heavier equipment
- **Dexterity:** Faster reload speeds, aiming down site speeds, switching between light equipment and weapons, faster looting speeds, better handling of lighter melee like blades
- **Agility:** Quicker movements, ability to quickly change directions (acceleration and deceleration improvements)
- Accuracy/Precision: Better
- **Combination Traits:** Power Attacks may require Strength + Dexterity levels, Power Movements (like a charge) may require Strength and Agility

Levelling up increases proficiency by reducing programmed handicaps, like weapon sway or reload speed, fatigue, balance disruptions (due to being disrupted with heavier equipment)

3 distinct non-player characters. For each character list a name (type), primary ability, attributes

Enemy Types all have varying attributes like
- Visual Perception: cone field of view to detect the player,

- Hearing: spherical range which may send enemies into an alerted state
- Smell: if patrolling, has lingering signals to the enemy types: temporal alert system that covers if the player moves through enemy patrol undetected (no hearing or sounds)

Enemy types will also have attributes to allow for

- Health, Strength, Endurance, Agility, Dexterity

Basic Zombie, has a basic melee attack

Armour/Heavy Zombie – stronger resistance against blades, weak against blunt

Big Zombie – Weak against large area attacks due to greater hitbox, like shotguns

Ranger Zombie, has a ranged attack like spitting acid – better with high precision ranged weapons

Alarm Zombie, when it detects the player, will scream to attract zombies

3 game objects. For each object list name (type), attributes (size, weight, durability, etc), rules (what does obtaining the object impart to the player or world?)

Firearms: e.g. Pistol, 1 inventory slot, 1.5 lbs, 100 durability, 7 rounds, 50 noise, 100 noise radius: player obtains a ranged weapon that they are able to use against zombies, effective against ranged enemy types

Melee Weapons, e.g. Baseball Bat, 1 inventory slot, 1.5 lbs, 40 durability, 5 noise (blunt is louder than blades), 5 noise radius: player obtains a melee weapon, more cost effective and silent option compared to firearms, with certain types of melee weapons being more effective against certain types of enemies

Armor, e.g. Sports Armor, 1 inventory slot, 2 lbs, 50 durability, 30% damage reduction: player gains

List 1 trap or secret game object. Describe the bonus, deficit, or other significant effect this object imparts on the player, npcs, or environment.

Biological mines that denotate with acid to debuff the player with vision loss, movement decrement and/or damage to the player.

List 2 types of trackable points that change during regular game play (currency, experience, etc.) other than the character attributes.

Money, experience and settlement resources will be used as trackable points.

Plan a marketplace by describing what types of game objects may be bought and/or sold.

Marketplace contains survivor items, like Ammunition, Weapons, Armor, Food/Nutrition Consumables,

State the win and loss conditions of your game.

The player wins when he reaches a number threshold for their settlement (e.g. 100 population), and loses when they lose all of their population

List potential resources to find professional looking graphics and sound assets for your game.

I plan on using assets generated by A.I., like through Image Generators and 3D model generators (Meshy AI).

List the way that the character will navigate the environment (physics needed, etc)

The character will navigate the environment using locomotion animations with translation to move the character object using the WASD keys, and mouse for camera view movement.

To support accessibility all Unity projects must incorporate customizable controls, use high contrast colour schemes, and visual cues for audio events.

All Unity projects must support some form of save and continue functionality.

All Unity projects must incorporate some form of difficulty levels (easy/hard minimum)

---End of Scope Document---

Functionality Chart

Feature	Notes	Player	Admin/Developer
Authentication System (Log In)	- Local username/password login	✓	✓
Authentication System (Reset Password)	- Local username/password only - Password strength checker - Secure password storage	✓	✓
Authentication System (New Account)	- Validate email address (confirmation link/code) - Real-time username availability check	✓	✗
Character Profile Creation	- Create character profile with attributes (level, exp, strength, etc.) - Allocate stat points upon leveling	✓	✗
Character Inventory System	- Persistent database for player inventory-Equip/unequip items - Weight limit based on strength	✓	✗
Marketplace System (Basic)	- Display items for sale (Ammunition, Weapons, Armor, Food) - Filtering based on type of item, level, durability, other characteristics - Buy/sell functionality	✓	✗
Marketplace System (Dynamic Pricing)	- Prices fluctuate based on game events (e.g., outpost attacks) - Backend logic for supply/demand	✓	✓
Safe Zone Mechanics	- Implement safe zones (outposts) with marketplaces - Provide utility locations (e.g., crafting stations)	✗	✓
Hostile Zone Mechanics	- Scavenge for supplies in hostile zones - Aggro system: attract zombies based on player actions	✓	✗
NPC Ecosystem (Basic Zombies)	- Basic Zombie: Melee attack - Ranger Zombie: Ranged attack (spit acid) - Alarm Zombie: Scream to attract others	✓	✗

	- Armour/Heavy Zombie – stronger resistance against blades, weak against blunt - Big Zombie – Weak against large area attacks due to greater hitbox		
NPC Ecosystem (Advanced)	- Dynamic NPC behavior (e.g., patrol routes, reactions to player) - Settlement population mechanics	✓	✗
Combat System (Basic)	- Weapon durability and maintenance cost - Shooting mechanics (accuracy, weapon sway)	✓	✗
Combat System (Skill-Based XP)	- Gain XP based on skillful performance (e.g., headshots) - Contextual XP gains (nutrition, health)	✓	✗
Level-Up System	- Reduce handicaps (weapon sway, reload speed) - Unlock new abilities	✓	✗
Game Objects (Weapons)	- Handgun: Stats (1 slot, 1.5 lbs, 100 durability) - Melee Weapons: Stats (1 slot, 1.5 lbs, 40 durability)	✓	✗
Game Objects (Armor)	- Sports Armor: Stats (1 slot, 2 lbs, 50 durability, 30% damage reduction)	✓	✗
Traps and Secrets	- Biological mines: Acid debuff (vision loss, movement decrement, damage)	✓	✗
Trackable Points System	- Money, experience, settlement resources - Update values dynamically during gameplay	✓	✗
Win/Loss Conditions	- Win: Reach settlement population threshold (e.g., 100) - Lose: Population drops to zero	✓	✗
Difficulty Levels	- Easy/Hard modes - Adjust enemy spawn rates, resource scarcity	✓	✗
Save and Continue Functionality	- Save game state to database - Load saved game	✓	✗
Accessibility Features	- Customizable controls - High contrast color schemes - Visual cues for audio events	✗	✗
Map/Terrain Design	- Hostile zones, safe zones, interaction zones	✗	✓

	- Terrain attributes (e.g., noise radius)		
Backend Game Logic	- Outpost attack logic (casualties, resource consumption) - Dynamic event system	✗	✓
Error Handling and Logging	- Handle errors gracefully - Log errors without exposing internal details	✗	✓

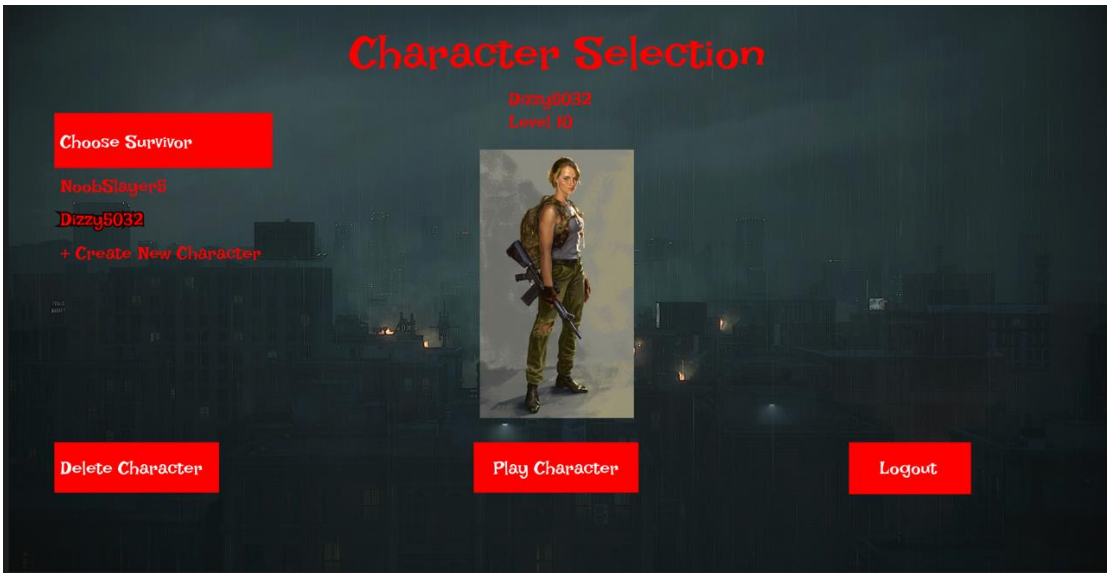
Screen Mockups and Program Flow Model Description

All screens are user screens, as games will require the administrator going into Unity to edit the game files for admin functionality.

The Login and Signup Screen: the user will be able to enter their credentials to login, the system will check if the login credentials are correct to login and will tell the user if the login credentials are incorrect if they are incorrect.

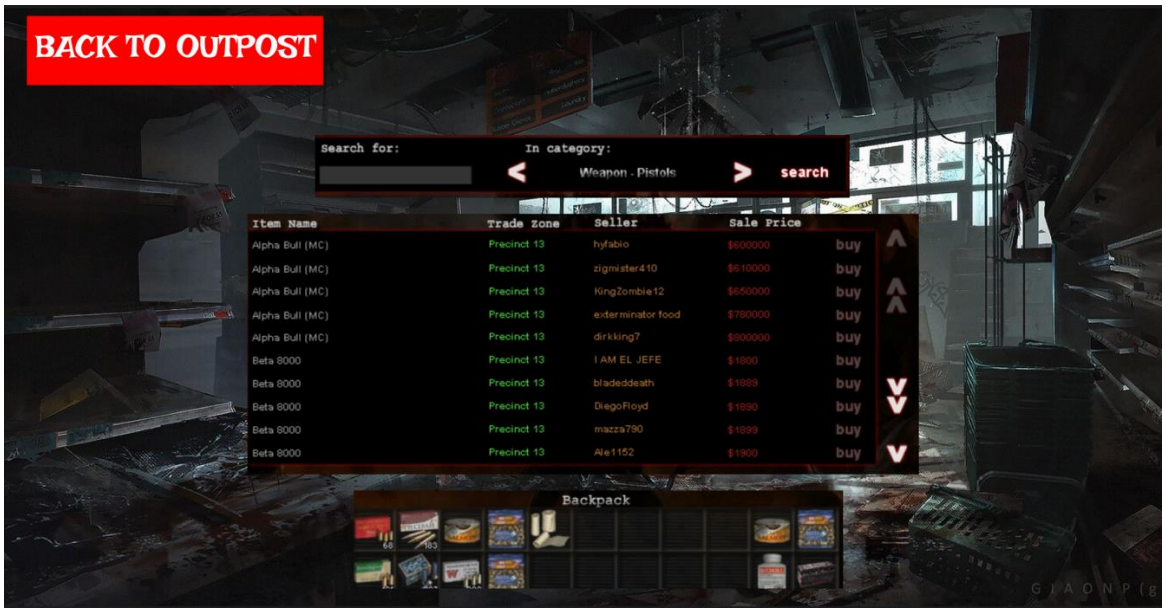
If the user enters a username and password and clicks signup, the system will check if a username exists with the inputted username, and will either tell the user that a user with that username exists if one exists already, or create a profile for the user if no username exists with the given username.

Character Selection Screen: Once the user has logged in, they will be able to create a new character. If they create a new character by clicking on "+ Create New Character", the "+ Create New Character" will become an empty field that they can type in a character name. The character name will be checked to see if it already exists, and if not hitting enter will create a level 1 character. Clicking on a character in the left under "Choose Survivor" will allow the user to select a character, which they can then proceed to enter the game with the selected character, or delete the character.

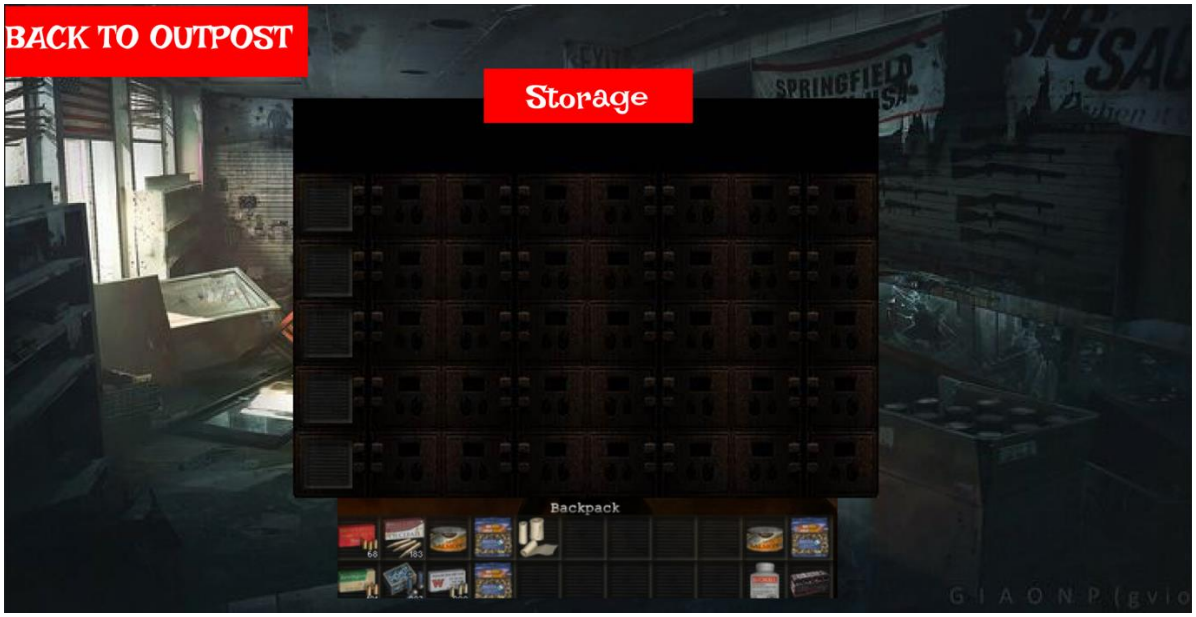


The Outpost Hub Screen (acts as the landing screen with outgoing links): This is the main hub area where the user can either navigate to their character overview, their inventory, the marketplace, into the inner city where the combat gameplay takes place, the character selection screen to select or manage characters, or logout.

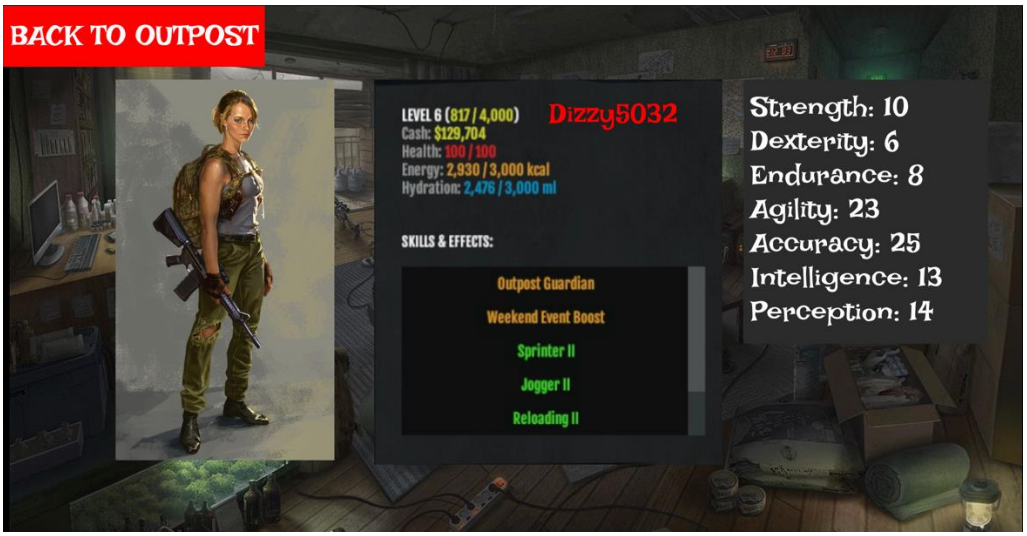
The Marketplace Screen: The marketplace consists of a search bar to filter for specific keyword and a category to filter for certain types of items, the selection of items available for sale which will display the item name, which outpost they are trading from, the seller, and the price of the item. Below the market is the players inventory which they will be able to put items into the marketplace to sell, and see items after they have purchased them. They will also be able to navigate to back to the outpost hub screen.



The Inventory and Storage Screen: The player is able to exchange items between their outpost storage and their inventory. They will also be able to navigate to back to the outpost hub screen.

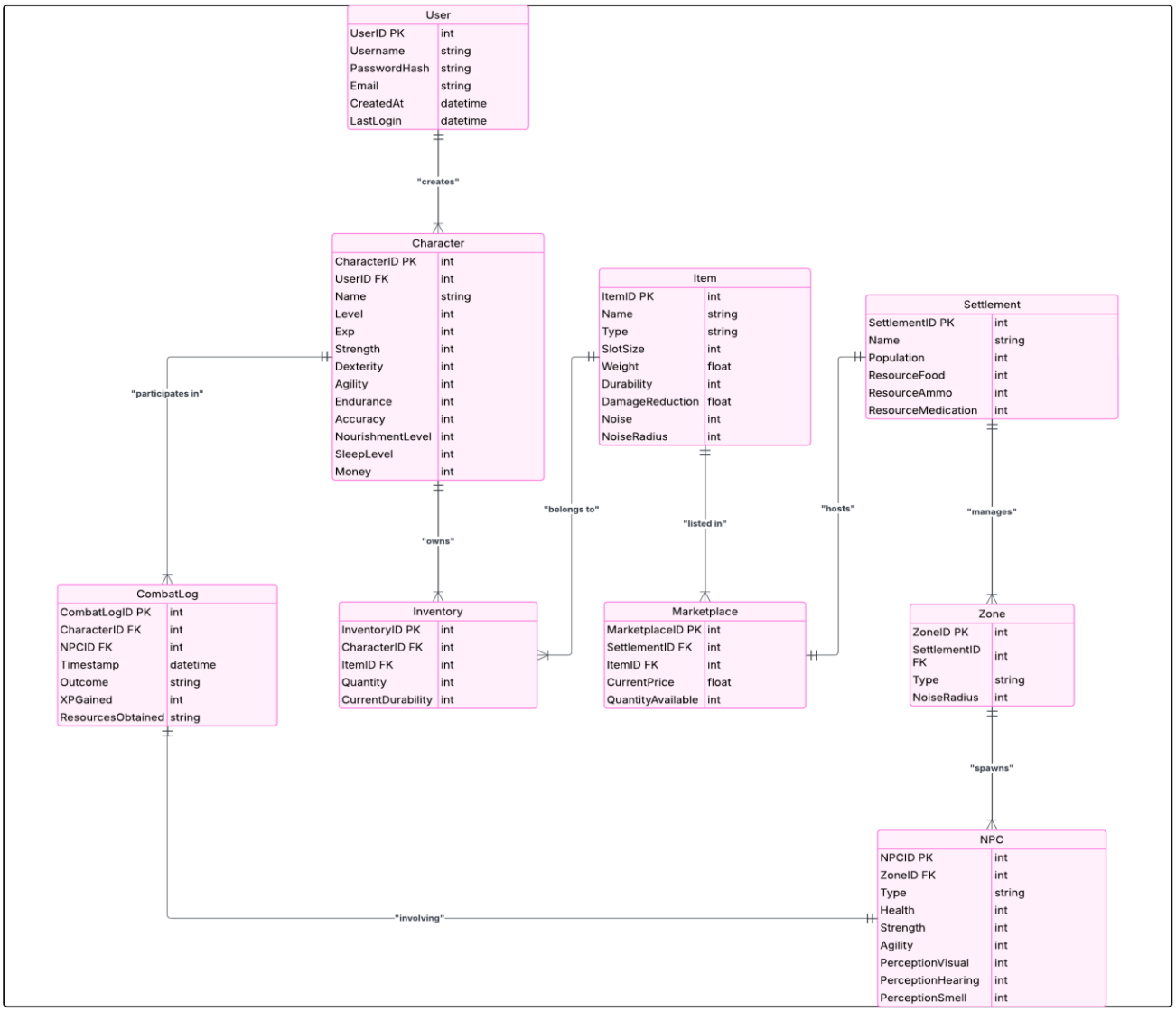


The Character Overview Screen: The character overview screen allows the user to check their character stats, including level, experience, cash, health, energy, hydration, skills & effects (etc., may vary with final design), as well as their characters attribute points. They will also be able to navigate to back to the outpost hub screen.



The Inner City/Gameplay example screen to show the Game UI: (Gameplay image is A.I. generated): The game is planned to be a third person over the shoulder survival horror shooter. Any missions appear along with their progress. The player must enter through the outpost gates in game to enter the outpost.

Database Schema (ERD)



Sample Data.

User Table

UserID	Username	PasswordHash	Email	CreatedAt	LastLogin
1	user1	(hashed value)	user1@email.com	2023-10-01 09:00	2023-10-05 14:00

Character Table

CharacterID	UserID	Name	Level	Exp	Strength	Dexterity	Agility	Endurance	Accuracy	Money
1	1	Survivor1	5	1000	10	8	7	12	15	500

Item Table

ItemID	Name	Type	SlotSize	Weight	Durability	DamageReduction	Noise	NoiseRadius
1	Pistol	Weapon	1	1.5	100	NULL	50	100
2	Sports Armor	Armor	1	2	50	30%	NULL	NULL

Inventory Table

InventoryID	CharacterID	ItemID	Quantity	CurrentDurability
1	1	1	1	85
2	1	2	1	45

Settlement Table

SettlementID	Name	Population	ResourceFood	ResourceAmmo	ResourceMedication
1	Outpost Alpha	50	200	150	100

Marketplace Table

MarketplaceID	SettlementID	ItemID	CurrentPrice	QuantityAvailable

1	1	1	250	10

Zone Table

ZoneID	SettlementID	Type	NoiseRadius
1	1	Safe	NULL
2	NULL	Hostile	50

NPC Table

NPCID	ZoneID	Type	Health	Strength	Agility	PerceptionVisual	PerceptionHearing
1	2	Basic Zombie	100	15	5	90	50

CombatLog Table

CombatLogID	CharacterID	NPCID	Timestamp	Outcome	XPGained	ResourcesObtained
1	1	1	2023-10-05 14:30:00	Win	50	Ammo:10

Normalization Notes:

- All tables are in 3NF (no transitive dependencies).
- Relationships are strictly 1-to-many (e.g., a user has many characters, a settlement has many zones).
- No many-to-many or 1-to-1 relationships.

This schema supports the game's functionality, including authentication, inventory management, dynamic markets, and combat tracking.

Test and Deployment Plan Description

1. Unit Testing

- **Purpose**: Verify individual components function correctly in isolation.
- **Tools**:
    - **Unity Test Framework**: For testing Unity-specific code (e.g., character movement, combat mechanics, UI interactions).
    - **Backend Framework**: Depending on the backend language, use Jest (JavaScript), PyTest (Python), or similar for API logic.
- **Examples**:
    - Test character stat calculations (e.g., strength affecting carry weight).
    - Test weapon durability degradation after use

2. Integration Testing

- **Purpose**: Ensure interactions between systems work as expected.
- **Tools**: Unity Test Framework for in-game integrations; Postman or Insomnia for API-client interactions.
- **Examples**: ...
    - Test that purchasing an item from the marketplace updates the character's inventory.
    - Test that combat outcomes (e.g., defeating an NPC) correctly update the database (CombatLog table).

3. Functional Testing

- **Purpose**: Validate that each major feature behaves as intended.
- **Approach**: Define test cases for successful behavior (test-to-pass) and failure conditions (test-to-fail), including boundary values. Automated testing is used where possible, supplemented by manual testing for Unity-specific features.
- **Major Features and Test Cases** (based on the Functionality Chart and Scope Document):
    - **Authentication System (Log In, Reset Password, New Account)**

        - *Test-to-Pass*: Valid username/password grants access; password reset with valid email sends a reset link; new account creation with unique username succeeds.
        - *Test-to-Fail*: Incorrect credentials deny access; duplicate username rejects signup; invalid email format fails validation.
        - *Boundary*: Empty fields, usernames/passwords exceeding 255 characters, special characters (e.g., !@#$%).
    - **Character Profile Creation and Management**
        - *Test-to-Pass*: Character created with default stats; stat points allocated on level-up.
        - *Test-to-Fail*: Duplicate character name rejected; stat allocation exceeds available points.
        - *Boundary*: Max/min stat values (e.g., strength = 0 or 100).
    - **Character Inventory System**
        - *Test-to-Pass*: Equip/unequip items; weight limit respects strength stat.
        - *Test-to-Fail*: Exceeding weight limit prevents item addition; unequipping non-equipped item fails.
        - *Boundary*: Weight at exact limit (e.g., strength = 10, weight = 10).
    - **Marketplace System (Basic and Dynamic Pricing)**
        - *Test-to-Pass*: Buy/sell items with sufficient funds; filter items by type; dynamic pricing adjusts after outpost attack.
        - *Test-to-Fail*: Insufficient funds block purchase; invalid filter returns no results.
        - *Boundary*: Buy item with exact funds (e.g., money = price).
    - **Safe Zone Mechanics**
        - *Test-to-Pass*: Player enters safe zone; marketplace accessible.
        - *Test-to-Fail*: Combat not possible in safe zone.
    - **Hostile Zone Mechanics**
        - *Test-to-Pass*: Scavenge supplies; aggro system triggers zombies on noise.
        - *Test-to-Fail*: No supplies outside hostile zones.
    - **NPC Ecosystem (Basic and Advanced)**
        - *Test-to-Pass*: Zombies attack based on type (e.g., Ranger Zombie spits acid); patrol routes adjust dynamically.
        - *Test-to-Fail*: NPCs ignore player outside perception range.
    - **Combat System (Basic and Skill-Based XP)**

        - *Test-to-Pass*: Weapon durability decreases; headshots grant bonus XP.
        - *Test-to-Fail*: Zero durability prevents weapon use; no XP without combat.
    - **Level-Up System**
        - *Test-to-Pass*: Level-up reduces weapon sway; new abilities unlocked.
        - *Test-to-Fail*: Level-up without sufficient XP fails.
    - **Game Objects (Weapons, Armor, etc.)**
        - *Test-to-Pass*: Pistol deals damage; armor reduces damage by 30%.
        - *Test-to-Fail*: Non-equipped items have no effect.
    - **Traps and Secrets**
        - *Test-to-Pass*: Biological mine applies acid debuff (vision loss, damage).
        - *Test-to-Fail*: Mine doesn't trigger outside its radius.
    - **Trackable Points System**
        - *Test-to-Pass*: Money, XP, and settlement resources update after actions.
        - *Test-to-Fail*: No updates without gameplay actions.
    - **Win/Loss Conditions**
        - *Test-to-Pass*: Population reaches 100 (win); drops to 0 (loss).
        - *Test-to-Fail*: Game continues between thresholds.
    - **Difficulty Levels**
        - *Test-to-Pass*: Hard mode increases zombie spawns; easy mode reduces them.
        - *Test-to-Fail*: Difficulty unchanged mid-game.
    - **Save and Continue Functionality**
        - *Test-to-Pass*: Game state saved and loaded correctly.
        - *Test-to-Fail*: Load fails without prior save.
    - **Accessibility Features**
        - See Accessibility Testing below.
    - **Map/Terrain Design**
        - *Test-to-Pass*: Hostile zones spawn enemies; safe zones don't.
        - *Test-to-Fail*: No crossover of mechanics (e.g., marketplace in hostile zone).
    - **Backend Game Logic**
        - *Test-to-Pass*: Outpost attack consumes resources; events adjust pricing.
        - *Test-to-Fail*: No changes without events.

    - **Error Handling and Logging**
        - *Test-to-Pass*: Errors logged without exposing details; user sees generic message.
        - *Test-to-Fail*: Internal data exposed on error (fail if occurs).

4. Security Testing

- **Purpose**: Protect against vulnerabilities like SQL injection and XSS.
- **Tools**: OWASP ZAP for automated scans; manual input testing.
- **Test Cases**:
    - **SQL Injection**: Input '; DROP TABLE Users; -- in username/password fields; expect sanitized input or error, not table deletion.
    - **XSS**: Input <script>alert('xss')</script> in fields (e.g., character name); expect no script execution in game or web components.
- **Metrics**: No unauthorized data access or code execution.

5. Accessibility Testing

- **Purpose**: Ensure usability for diverse players, per Unity project requirements.
- **Tools**: axe DevTools, manual testing.
- **Test Cases**:
    - **Customizable Controls**: Remap WASD to arrow keys; verify functionality.
    - **High Contrast Color Schemes**: Enable high contrast; verify improved visibility (e.g., text readable on dark backgrounds).
    - **Visual Cues for Audio Events**: Zombie scream triggers visual alert (e.g., screen flash); verify presence.
- **Metrics**: All controls remappable, color scheme toggle effective, 100% audio events with visual cues.

6. API Testing

- **Purpose**: Verify REST API endpoints function correctly.
- **Tools**: Postman or Insomnia.
- **Expected APIs** (based on features; documentation links TBD upon development):
    - /login: Authenticate user.
    - /characters: Create/manage character profiles.
    - /inventory: Update/retrieve inventory.
    - /marketplace: Buy/sell items, fetch prices.

    - /combat: Log combat outcomes.
- **Test Cases** (for each endpoint):
    - *Valid Input*: Correct data returns 200 OK with expected JSON.
    - *Invalid Input*: Malformed data returns 400 Bad Request.
    - *Response Time*: < 2 seconds under normal load.
    - *Error Handling*: Unauthorized access returns 401; server errors return 500 with generic message.
    - *Authentication*: Valid token required; invalid token returns 403.
    - *Data Integrity*: Database reflects changes (e.g., item quantity decreases after purchase).

Notes

- Testing is ongoing throughout development, with automated tests run via Unity Test Runner or CI/CD pipelines (e.g., GitHub Actions).
- Functionality testing occurs on the hosted system post-deployment.
- Performance, cross-platform, and cross-browser compatibility testing are not required (Chrome/Edge only).

Deployment Plan

The deployment plan outlines the hosting solution for "Dizzy's Disease," ensuring a commercially available platform is used (self-hosting is not supported). The game consists of a Unity WebGL client and a backend API with a database, deployed as follows:

1. Hosting Solution

- **Unity WebGL Client**:
    - **Platform**: Netlify.
    - **Reason**: Ideal for static content like WebGL builds; offers a free tier, easy deployment, and subdomain support. The game runs in Chrome/Edge browsers as a WebGL build, aligning with requirements.
- **Backend API and Database**:
    - **Platform**: Heroku.
    - **Reason**: Supports REST API hosting and includes Heroku Postgres (a managed PostgreSQL database), simplifying setup. CSUNIX was considered,

but its "localhost only" restriction complicates API hosting, so a fully cloud-based solution is preferred.
    - **Setup**: API server (e.g., Node.js, Python) connects to Heroku Postgres; no direct database connections are used, adhering to REST API requirements.

2. Estimated Cost (3 Months)

- **Netlify Free Tier**:
    - 100GB bandwidth/month, 300 build minutes/month.
    - Cost: $0 (sufficient for a capstone project).
- **Heroku Free Tier**:
    - 550 dyno hours/month (API server); Heroku Postgres free tier (10,000 rows limit).
    - Cost: $0 (adequate for development/testing; assumes < 10,000 rows).
- **Total**: $0 over 3 months, assuming usage stays within free tiers. If exceeded (e.g., database rows > 10,000), Heroku Postgres costs $9/month, but this is unlikely for a prototype.

3. Access Method

- **Subdomains**:
    - Game: https://dizzys-disease.netlify.app.
    - API: https://dizzys-disease-api.herokuapp.com.
- **Reason**: Both platforms provide consistent subdomains, avoiding dynamic IPs as required. No custom domain registration is needed.

Notes

- The WebGL build is deployed via Netlify's drag-and-drop or CLI tools.
- The backend API is deployed via Heroku's Git integration, with environment variables for database credentials.
- The solution supports the project's needs: a browser-based game with persistent data via a REST API and SQL database.

Development Timeline

Development Timeline for "Dizzy's Disease"

Milestone	Start Date	Tasks Planned
0	2025-09-03	- Review proposal, finalize frameworks (Unity, Netlify, Heroku, PostgreSQL)
		- Install needed tools, IDEs (Unity Editor, Visual Studio, PostgreSQL client), configure dev environment
		- Upload approved proposal and framework documents to GitHub repository
1	2025-09-08	- Set up Unity project, configure GitHub repository with initial commit
		- Implement basic authentication: login and signup with secure password hashing
		- Create User table in PostgreSQL database
		- Commit initial project setup and authentication code
	2025-09-15	- Implement password reset with email validation (mock email system for demo)
		- Develop character creation: create character with default stats (strength, dexterity, etc.)
		- Implement character selection screen UI in Unity
		- Integrate with Character table in database
		- Test authentication and character creation, commit changes
	2025-09-22	- Create and submit progress report and demo video (Milestone 1)

		- Start inventory system: design inventory UI, add items functionality
		- Implement equip/unequip mechanics
		- Update database with Inventory table
		- Commit inventory progress
2	2025-09-29	- Complete inventory system: implement weight limits based on strength stat
		- Begin marketplace: display items for sale with basic buy functionality
		- Update database with Item and Marketplace tables
		- Commit inventory and marketplace changes
	2025-10-06	- Implement sell functionality in marketplace
		- Add filtering to marketplace (by type, durability)
		- Start map/terrain design: create safe zones in Unity
		- Implement safe zone mechanics (marketplace access, no combat)
		- Commit marketplace and safe zone progress
	2025-10-13	**Break week** - Catch up on any pending tasks or fix issues
	2025-10-20	- Deploy initial progress to hosting platform (Unity WebGL on Netlify, backend API on Heroku)
		- Create and submit progress report and demo video (Milestone 2)
		- Begin hostile zone mechanics: implement scavenging system
		- Implement Basic Zombie NPC with melee attack
		- Update database with Zone and NPC tables
		- Commit deployment and hostile zone progress
3	2025-10-27	- Develop aggro system for zombies (noise-based attraction)
		- Implement Ranger Zombie with ranged attack (acid spit)
		- Start combat system: player attacks, zombie attacks, damage calculation
		- Update NPC table with additional zombie types
		- Commit combat and NPC progress
	2025-11-03	- Complete combat system: weapon durability, accuracy based on stats
		- Implement level-up system: stat allocation, reduce handicaps (e.g., weapon sway)
		- Add Alarm Zombie with scream mechanic
		- Update database with CombatLog table
		- Commit level-up and combat enhancements

	2025-11-10	- Create and submit progress report and demo video (Milestone 3)
		- Implement dynamic pricing in marketplace based on game events (e.g., outpost attacks)
		- Add settlement mechanics: track population and resources
		- Implement win/loss conditions (population 100 to win, 0 to lose)
		- Commit settlement and marketplace updates
4	2025-11-17	- Add save and continue functionality: save game state to database
		- Implement accessibility features: customizable controls, high contrast mode, visual audio cues
		- Finalize all features: add difficulty levels (easy/hard), traps (biological mines), error handling
		- Conduct thorough testing and debugging
		- Commit final features and fixes
	2025-11-24	- Write the final project report
		- Prepare the presentation video
		- Ensure all deliverables are ready for submission
	2025-12-01	- Create and submit final report and presentation video (Milestone 4)
		- Upload all needed deliverables to LMS
		Link to Capstone GitHub Repository: `https://github.com/Steve-at-Mohawk-College/capstone-project-Jinphinity`
		Final Hosted Project Due: December 5th, 2025 - no extensions

Explanation of the Timeline

Key Constraints Addressed

- **Development Period:** Spans Weeks 2 to 12 (September 8th to November 23rd), with Week 1 reserved for setup and Week 14 for final submission. Week 13 is dedicated to writing the final report, as required.
- **Deployment by Milestone 2:** The initial deployment to Netlify (Unity WebGL) and Heroku (backend API) is scheduled for October 20th, 2025 (Week 8), ensuring a functional prototype is hosted by Milestone 2.
- **Front-Loaded Tasks:** Weeks 2-4 (September 8th to September 28th) include critical foundational tasks: project setup, authentication, character creation, and inventory system initiation, aligning with the guideline to allocate more work early.

- **Break Week:** Week 7 (October 13th-19th) is reserved for catch-up, with no new tasks scheduled.
- **1-Week Intervals:** All tasks are broken into weekly segments for alignment with project meetings.
- **GitHub Commits:** Regular commits are planned each week to reflect ongoing progress with meaningful messages.
- **Fixed Milestones:** Milestone deliverables are scheduled as pre-filled in the template: September 22nd (Milestone 1), October 20th (Milestone 2), November 10th (Milestone 3), and December 1st (Milestone 4).

Feature Implementation

The timeline distributes the major functionality items from the Functionality Chart and Scope Document across the 12 weeks, ensuring a logical progression:

- **Weeks 2-3:** Core systems (authentication, character creation) to establish the player foundation.
- **Weeks 4-5:** Inventory and marketplace basics for item management and economy.
- **Weeks 6-8:** Map/terrain, safe/hostile zones, and initial NPC/combat for gameplay loop, culminating in deployment.
- **Weeks 9-11:** Enhanced combat, NPCs, level-up, and settlement mechanics for depth and win/loss conditions.
- **Week 12:** Final features (save functionality, accessibility, difficulty) and testing for polish.

Deployment and Testing

- **Deployment:** By Week 8, a basic version with authentication, character management, inventory, marketplace, and simple gameplay is deployed, allowing early issue identification.
- **Testing:** Integrated throughout, with unit tests (Unity Test Framework, backend framework), integration tests (Postman), and functional tests conducted weekly, finalized in Week 12.

This timeline ensures "Dizzy's Disease" is developed systematically, meeting all requirements while allowing flexibility for adjustments during the break week. Regular commits and milestone deliverables provide checkpoints for progress evaluation.

Reflection on Capstone Proposal

Preparing the proposal for my capstone project, "Dizzy's Disease," before diving into coding has proven to be an immensely valuable experience. This planning phase provided me with structure, clarity, and a strong foundation that I am confident will streamline the development process. As I reflect on the various components of the proposal—such as the development timeline, test and deployment plan, database schema, screen mockups, and functionality chart—I can pinpoint several elements that stood out as particularly useful. These deliverables not only shaped my approach to the project but also ensured that I was well-prepared before writing a single line of code. Below, I'll discuss the aspects I found most beneficial, why I'm glad I tackled them early, and touch on any deliverables that felt less impactful, as well as my thoughts on the course structure.

Most Useful Elements of the Proposal

Development Timeline

The development timeline was one of the most critical components of the proposal. Breaking down the 12-week development period (from September 8th to November 23rd) into specific tasks—like setting up authentication and character creation in Weeks 2-3, building inventory and marketplace systems in Weeks 4-5, and finalizing save functionality and accessibility in Week 12—gave me a clear roadmap. This structure ensured that I front-loaded foundational tasks, such as player authentication and character management, which are essential for the game's core mechanics. It also included a break week in Week 7 for catch-up, which I appreciate as a buffer for unexpected delays. I'm glad I created this timeline before coding because it helped me visualize the project's progression, manage

my time effectively, and reduce the overwhelm of tackling a large-scale project. Without it, I might have jumped into coding haphazardly, risking wasted effort on features out of sequence.

Test and Deployment Plan

The test and deployment plan was another standout deliverable. Outlining how I would verify the game's functionality through unit testing (e.g., using Unity Test Framework for character stat calculations), integration testing (e.g., ensuring marketplace purchases update the inventory), and functional testing (e.g., validating win/loss conditions) forced me to think about quality assurance from the start. Planning the deployment—hosting the Unity WebGL client on Netlify and the backend API with Heroku Postgres by Week 8—also ensured I had a concrete strategy for making the game accessible. I'm thankful I did this early because it encouraged me to prioritize security (e.g., testing for SQL injection and XSS) and accessibility (e.g., customizable controls and high-contrast schemes) upfront. This preparation will likely save me from last-minute fixes and result in a more polished, user-friendly application.

Database Schema

Designing the database schema was a game-changer for the backend of "Dizzy's Disease." Creating tables for users, characters, items, inventories, settlements, marketplaces, zones, NPCs, and combat logs—normalized to 3NF—gave me a clear picture of how data would flow through the game. For instance, linking the inventory table to the character and item tables ensured I could efficiently track what each player carries, while the marketplace table supported dynamic pricing based on game events. I'm glad I finalized this before coding because it provided a solid foundation for implementing persistent features like character inventories and settlement resources. Without this upfront work, I might have faced significant refactoring later when integrating the database with the game logic.

Screen Mockups and Program Flow Model

The screen mockups and program flow model were invaluable for visualizing the user experience. Sketching out screens like the login/signup interface, character selection, outpost hub, marketplace, and inner city gameplay helped me design an intuitive UI. For example, planning the outpost hub as a central navigation point with links to inventory, character overview, and combat zones ensured a cohesive flow. I'm grateful I did this early because it allowed me to spot potential usability issues—like ensuring navigation was straightforward—before coding the interface. This preparation will help me stay aligned with my intended design and minimize costly UI revisions down the line.

Functionality Chart

The functionality chart, detailing features like authentication, RPG mechanics (e.g., leveling up stats like strength and dexterity), marketplace dynamics, and win/loss conditions (e.g., settlement population reaching 100 or dropping to 0), provided a comprehensive checklist of what the game needed to deliver. It helped me prioritize tasks—like implementing core systems before advanced NPC behaviors—and ensured I didn't overlook critical requirements. I'm glad I completed this before coding because it clarified the project's scope and kept me focused on delivering a complete survival-RPG experience. It also aligned my efforts with the capstone's expectations, making the development process more manageable.

Less Valuable Deliverables

While most of the proposal felt essential, there were moments when certain aspects seemed less high-value at this stage. For instance, documenting the exact mechanics of the NPC ecosystem (e.g., how Ranger Zombies spit acid or Alarm Zombies attract others) or the level-up system's finer details felt a bit premature, as these elements might evolve during playtesting. Spending time on such specifics occasionally felt like over-planning when I wasn't yet sure how they'd play out in practice. That said, I recognize that this level of detail was intended to force thorough consideration of all game components, and even these efforts contributed to a stronger overall plan. In retrospect, I wouldn't skip them entirely, but perhaps streamline them to focus on broader concepts rather than minute specifics at this stage.

Thoughts on Course Structure

The course provided examples for each deliverable—like the development timeline, test plan, or functionality chart—to show what the professor expected us to submit. However, those examples were all tailored to web apps, and they didn't translate well to a Unity game like "Dizzy's Disease." For instance, the sample test cases were focused on web app features, such as login forms or shopping carts, which didn't match up with game-specific elements like character stats or combat systems. Similarly, the program flow examples were built around web navigation, not the interactive UI you'd find in a game. This mismatch made it harder to understand how to apply the requirements to my project. It would've been more helpful if the examples included something Unity-specific to make the expectations clearer for game development. I figured out how to adapt them eventually, but the web app focus definitely added some unnecessary guesswork.

Promise to Future Self

As I begin developing "Dizzy's Disease" and approach graduation, I promise to maintain a disciplined and proactive mindset. Specifically, I will stick to the development timeline I've crafted, working incrementally each week and avoiding procrastination, even when tasks feel daunting. I'll also commit to seeking help promptly—whether from peers, professors, or online communities—rather than letting pride or frustration slow me down. Finally, I'll prioritize my well-being by balancing work with rest, exercise, and time with friends, ensuring I don't burn out before the finish line. By honoring these commitments, I'm setting myself up to deliver a capstone I'm proud of and transition confidently into my career.