Basically, all my goals set for this final project is accomplished, accompanied by some extra functions added. Below is a summary of all features I have done for this final project:

**Basic Goals:**

* **Procedural Map Generation**
  + Graph structure for the map
  + A\* Algorithm used for enemy pathfinding / optimizing map generation logics
* **Dynamic Map Connection**
  + The Map changes dynamically over time. Each time you traverse a node, 15 minutes will be passed. The more time you passed, the more chance for a specific road to be “broken”, and it will be recovered some point in the game
  + Also used A\* to make sure once a road is broken, the player is still able to traverse till the end of the map (for now, the “end” of the map is the topmost nodes)
* **Procedural Road Generation**
  + Road models are procedurally generated according to the data of nodes.
* **Zombie Dynamic Movement On Map**
  + The enemies also move on the map over time.
  + Logics:
    - If the enemy is far away from the player, just move randomly
    - If the player is close to the enemy, that enemy will use A\* algorithm to try to approach the player
* **Card Structures / Procedural Card Generation / Procedural Card Description Generation**
  + Five types of cards \* 4 Rarity
    - Area / Attack / Armor / Skill / Character Cards
  + The effects of each card are not coded in their corresponding types, but each effect is written in a separate class (Design Pattern: “Command Pattern”)
  + This enables me to make Procedurally generated cards.
  + Each effects also have its own type and rarity (See “EffectClassificationModel” Class). When a Procedurally Generated Card of a specific type and rarity is required, the generator will only pick effects that matches the type & rarity requirements.
    - Also make sures: No two “EffectContainers” (effects that can execute other effects) in a same card; No two “Pointable” effects (Effects that require the player to “point” to a specific enemy to execute)
  + Moreover, each effects themselves are also not “fixed”. In other words, some variables in some effects are not hardcoded (like “draw X cards”). They can be randomly generated when they are attached to a procedurally generated card. And their Cost Value is also calculated according to their strength.

**Extra Functions**

* **Visualized Card Editor – How to Use**
  + In “CardConfigureScene”, you can edit and create cards in a visualized editor:
    - Open the inspector page of “CardConfigure/CardConfigureTemplate“.
    - To Edit a Card:
      * Input a card Id (from 0-15)
      * Click “Read Existing Card by ID”
      * Then, the card info will be loaded, and you can make changes to it, then click “Save Card Properties”
    - To Create a Card
      * Simply input a non-existing card ID
      * Input the rest of the information
    - **Edit the Position and Size of Card Illustration**
      * This visualized card editor allows you to edit the position / size of a card’s illustration in a much easier way than hard-code it
      * After you assign a sprite to “Card Illustration Sprite”, the card in the Scene will have that illustration.
      * To change the position and size of that card, simply change the position and scale of the “CardIllustraion” object under “CardIllustrationMask”. Then “Save Card Properties”
      * After that, in the game, the card’s illustration’s size and position will reflect your change
    - To view a list of all cards in the game, go to Assets/CardProperties object. (it’s a ScriptableObject so you can edit it via the Inspector Window)
* City Generation
  + As shown in the map :) – no more explanations!
* Multilanguage Support
  + On the top-bar of the game, you can switch languages.
  + The only 3 languages supported by the game are: English, Simplified Chinese, Traditional Chinese (Which are only languages that I master :(
  + The card description will match the language (if you are “Viewing Deck”, you may need to exit that panel then reenter it to see the change)
* **Seed System / Save and Load**
  + I implemented a seed system in the game (“pseudo random”)(See “SeedSystem”) in the code.
  + The Seed System contains three Random Generators, each are used for Map Generation, Battle AI Random Generation (not been used for now), and General Random Generator.
  + Each time the game saves, the game will serialize these random generators, as well as all the other states of the game (like passed levels, enemy states, game time, etc.).
  + As you reenter the game, everything will be restored, including the three random generators

**Core Codes:**

1. About “MikroFramework”: This is a game framework based on MVC Architecture that I made last year. My code is basically all based on this architecture. For more info, see my GitHub: <https://github.com/cty288/MikroFramework>
2. Core Map Generation and A\* Code: <https://github.com/cty288/CardGame/tree/master/Assets/05.%20Scripts/Systems/Map>
3. Card Data Structures Code: <https://github.com/cty288/CardGame/blob/master/Assets/05.%20Scripts/Model/Game/Cards/BaseCardInfo/CardInfo.cs>
4. Card Effects Base Code: <https://github.com/cty288/CardGame/blob/master/Assets/05.%20Scripts/Model/Game/CardEffectCommands/EffectCommand.cs>
5. Reward System and Procedural Card Generation Core Code:
   1. Reward System and Procedural Card Generation Algorithm: <https://github.com/cty288/CardGame/blob/master/Assets/05.%20Scripts/Systems/RewardSystem/RewardSystem.cs>
   2. All Possible Effects: <https://github.com/cty288/CardGame/tree/master/Assets/05.%20Scripts/Model/Game/CardEffectCommands>
   3. Procedurally Generated Card Base Class: <https://github.com/cty288/CardGame/blob/master/Assets/05.%20Scripts/Model/Game/Cards/ProceduralNormalCard.cs>
6. Visualized Card Editor: <https://github.com/cty288/CardGame/blob/master/Assets/05.%20Scripts/Utilities/CardConfigure.cs>