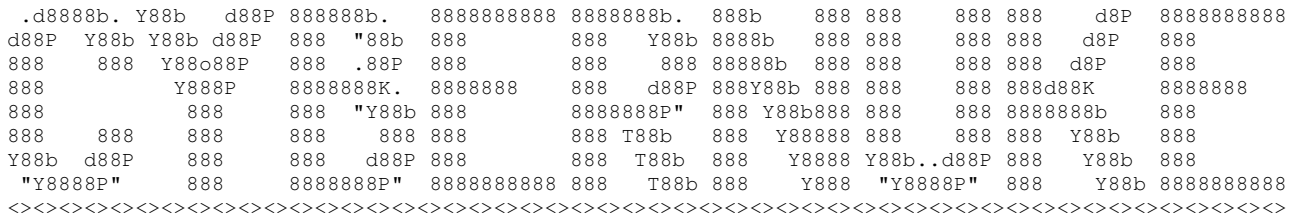

CYBERNUKE

POST-APOCALYPTIC CYBERPUNK ROGUELITE

J. Kenneth Wallace, Vance Brender-A-Brandis

Abstract

CYBERNUKE is a roguelite game developed in the C# WPF framework. You play as an amnesiac cyborg who awakens in the middle of a destroyed futuristic city and must recover their memories. The game combines a post-apocalyptic and cyberpunk setting to create a unique atmosphere not previously explored by other games. The setting of the game is inspired by popular stories such as *Fallout*, *Cyberpunk 2077*, and other game series like *Dark Souls* and *Megami Tensei*.

1. Introduction

For this project we aim to create a video game to gain experience with C#, WPF, and game design. We called the game "CYBERNUKE," combining the game's cyberpunk and post-apocalyptic elements into a single title.

The setting of the game is rooted in the cyberpunk genre, taking place in a massive futuristic city where buildings as tall as the Burj Khalifa are commonplace, gangs are armed with high-tech laser guns, and the wealth gap is at its widest. On a peaceful day in the twenty-first century, the city is attacked by an unknown power, which is followed by a nuclear explosion and the displacement of the entire city into another dimension. The story's cyberpunk portion is inspired by the game *Cyberpunk 2077*, while the post-apocalyptic portion is inspired by *Fallout* and other post-apocalyptic games.

CYBERNUKE's gameplay is influenced by games such as the *Shin Megami* and *Dark Souls* series. Shin Megami inspired the game's turn-based combat, while Dark Souls inspired the overworld map and exploration. Moreover, CYBERNUKE is heavily influenced by the roguelike genre, particularly games like *Slay the Spire*, *The Binding of Isaac*, and *FTL: Faster Than Light*.

We hope that those who enjoy the gameplay and story of the works that inspired CYBERNUKE will enjoy the game we make as well.

1.1. Audience

CYBERNUKE has a diverse setting, story, and gameplay that draws inspiration from a wide range of games and stories. We want to draw in fans by fusing two sizable genres—cyberpunk and post-apocalyptic—and putting it into a roguelite game. Those looking for gameplay should expect turn-based combat and a top-down world filled with exploration and encounters akin to *Pokémon*. Those who come for the story will be treated to a mysterious tale about someone piecing together their memories in a new, hostile cyberpunk hell.

1.2. Background

CYBERNUKE is inspired by roguelike games, but it does not completely follow them, making CYBERNUKE a rogue-*lite*. Roguelite games are a subgenre of roguelike games. The Berlin Interpretation of a "roguelike," developed at the 2008 International Roguelike Development Conference, is the most widely accepted definition. It specifies that all roguelikes must have the following characteristics: [3]

1) Permadeath: the player has one life per game.
2) Random environment generation: no map or encounter is ever the same.
3) Exploration and discovery: the player explores the map and finds items.
4) Turn-based, grid-based, non-modal gameplay: all actions are turn-based, on a grid map, on a single screen that holds all windows and doesn't change.
5) Hack-n-slash: the player defeats a lot of monsters.
6) Resource management: the player manages a limited amount of resources.

Permadeath, random environment generation, and non-modal gameplay are not included in CYBERNUKE. As such, our game is classified as a roguelite because it lacks most of the required features. Specifically, roguelite as used by CYBERNUKE means the player has infinite lives, explores a pre-built map with loot spread out, encounters enemies at fixed rates, fights in turn-based combat, and manages a limited amount of resources.

CYBERNUKE's setting is a hybrid of two genres: cyberpunk and post-apocalyptic. Cyberpunk is defined as a *"science-fiction subgenre characterized by countercultural antiheroes trapped in a dehumanized, high-tech future,"* [1] or a dystopian future full of holograms, lasers, and crime. The post-apocalyptic genre is defined by the collapse of society as a result of a disaster, with a few survivors forming small civilizations. The combination of these two genres serves as the foundation for the rest of the story and gameplay.

1.3. Impacts

If enough effort is put in, CYBERNUKE has the potential to amass a sizable fan base. An optimistic estimate would be that CYBERNUKE reaches a large audience that plays, discusses, and enjoys the game in general. The game would then inspire others to create stories or games in the same hybrid cyberpunk-post-apocalyptic genre. A more realistic estimate would be that the game is simply released and is only played by a small number of people. Expecting too much from such a small game is unrealistic.

1.4. Challenges

Three big challenges will need to be overcome in this project:

1) Inexperience in C# and WPF
2) Clear communication of ideas and implementations
3) Beating the time limit
4) How to design a game

We've both never worked with the WPF framework, let alone C#. As the project progresses, we will have to learn how to work with both. This can be mitigated by reading documentation or watching tutorials.

Clear communication can also be a big issue and can stop work if people aren't on the same page. What makes this worse is that we are both inspired by different games which makes it difficult to communicate ideas of one game that the other person hasn't experienced. The clear solution to a lack of clear communication is using standardized methods of communication like UMLs and Diagrams. UML and Discord will definitely be the work-horses of communication throughout the project.

Another issue is the project's time frame. Having to not only learn but also implement a large number of designs in a short period of time will be stressful, but it should be possible.

The most serious issue is that we have no experience designing games. There is a significant difference between playing and creating a game, which will need to be addressed as we continue to work.

2. Scope

In order of importance, these are the fundamental game mechanics that must be implemented for *CYBERNUKE* to be considered functional:

1) Operational Interfaces (Main Menu, Combat Menu, Town Menu, etc.)
2) Menu-switching Functionality
3) Secondary Interfaces (Character Menu, Inventory Menu)
4) Enemies
5) Overworld Maps and Map Rendering
6) Overworld Map Functionality (Character Movement, Interaction, Enemy Encounters)
7) Combat Implementation
8) Player Saves & Load Menu

1. The Operational Interfaces are the menus that the player uses to interact with the game. (Main Menu, Combat Menu, Town Menu, Overworld Menu, Cutscene Menu).
2. The ability to switch between Menus.
3. The Secondary Interfaces are other menus that are accessed within other menus. (Character Menu, Inventory Menu).
4. Enemies and Enemy Data.
5. The ability to create Overworld maps and render them onto the screen.
6. The ability to move a player's character while they are on the map and the ability to interact with parts of that map as the player.
7. Enemies are the monsters/humans that the player will encounter throughout the game. Enemy Encounters refers to how you will find monsters in the first place, which is primarily done through a "percent encounter" system akin to the old *Pokemon* games. There may also be opportunities for pre-scripted encounters.
8. Combat refers to fights between the player character and enemies in this game, which will be turn-based. Actual combat is made possible by the Combat Interface.
9. Players can save their state (at specific points in the game) and load back in at a later time using Player Saves and the Load Menu.

After the fundamental game mechanics are in place, additional features that flesh out the game can be added.
Possible stretch-goal features (In no particular order):

1) Companions and Party System
2) Hacking Puzzles
3) More Enemy Types
4) More Story Content
5) Items, Loot, Weapons, Armor, etc.
6) Cybernetic Augmentation
7) Leveling System
8) Currency and Merchants
9) Expanded Character Creation (Species, Backstory, etc.)

2.1. Requirements

The required mechanics for the game to function at a basic level are defined in the first part of the scope. These mechanics are required as they are the foundation of the game's most basic actions such as traversing menus, moving around the map, viewing your character's stats, and fighting enemies.

The non-required mechanics for the game are defined in the second part of the scope. These mechanics are not required for the game to function, but are for fleshing the game out into a more playable state. They build upon the foundation laid by the functional mechanics to create something more than a roguelite mockup.

2.1.1. Functional.

- User needs to be able to start a new game as well as load a previous game if one exists.
- Additionally, User must be able to save their game while playing
- User must be able to move around the game world - both on normal map and overworld map
- User must be able to participate in combat by performing actions like attacking, guarding, using items, etc.
- User must be able to interact with NPCs, like for shopping or for progressing the story
- User must be able to manage inventory - dropping or equipping/using items, using quest items, etc.

Use Case ID	Use Case Name	Primary Actor	Complexity	Priority
1	Start New Game	Player	Low	1
2	Save Game	Player	Low	1
3	Load Game	Player	Low	1
4	Movement	Player	Med	1
5	Combat	Player	Med	1
6	Talk to NPCs	Player	Low	3
7	Manage Inventory	Player	Low	3
8	Check Character Stats	Player	Low	3
9	Manage Companions	Player	Med	1

TABLE 1. CYBERNUKE USE CASE TABLE

- User must be able to see and examine their character's stats - each stat should have a description of what it affects
- User must be able to manage companions - change order, dismiss or recruit, check equipment and skills of, etc.

2.1.2. Non-Functional.

- Autosaves - User should have automatic saving in case of crashing, failing battles, and general convenience
- Simple Combat AI - Enemies should be able to attack, use skills, etc. during battle through use of algorithms or basic attack sequences
- Shop Stock - Basic inventory stocked so User can buy various items at different times (can reset shop every nth battle if we want)
- Map autoscrolling - Camera should center on User and map should scroll as User moves around
- I didn't know what else to put here.

2.2. Use Cases

This subsection is arguably part of how you define your project scope (why it is in the Scope section...). In a traditional Waterfall approach, as part of your requirements gathering phase (what does the product actually *need* to do?), you will typically sit down with a user to develop use cases.

You should have a table listing all use cases discussed in the document, the ID is just the order it is listed in, the name should be indicative of what should happen, the primary actor is typically most important in an application where you may have different levels of users (think admin vs normal user), complexity is a best-guess on your part as to how hard it should be. A lower number in priority indicates that it needs to happen sooner rather than later. A sample table, or Use Case Index can be seen in Table 1.

Use Case Number: 1
Use Case Name: Start New Game
Description: A player has opened the game and wishes to start a new save. They will click on a "New Game" button. This will start the process to start a new game.

1) Player opens game to the Main Menu
2) Player left-clicks on "Start Game" button.
3) A new save is created and the Player is shunted into the game world

Termination Outcome: A new game will be started for the Player

Use Case Number: 2
Use Case Name: Save Game
Description: Player has achieved a goal or wishes to save their progress before stepping away. They will click on a "Save Game" button. This will kick off a process to record map position, inventory, character stats and companions, etc. etc.

1) Player opens the pause menu (presumably with ESC key).
2) Player left-clicks on "Save Game" button.
3) The Player's progress and status is recorded into a file and saved to a dedicated "saves" folder

Termination Outcome: A new saved game is recorded and exported to a "saves" folder for future loading

Use Case Number: 3
Use Case Name: Load Game

Description: Player has died or has opened the game and wishes to load their previously saved progress. They will click on a “Load Game” button. This will kick off a process to load their saved status and start from where they left off.

1) Player opens the game
2) Player left-clicks on ”Load Game” button
3) Player chooses from a list of saves from within their ”saves” folder

Termination Outcome: Player loads their saved game and is returned to the point where they saved their game with their status, equipment, stats, etc. the same as when they saved.

Alternative: Player is within the game world and is alive

1) Player opens the pause menu
2) Player left-clicks on ”Load Game” button
3) Player chooses from a list of saves

Termination Outcome: Player loads their saved game from within the game world and is returned to the point where they saved their game with all faculties the same as when they saved

Alternative: Player is within the game world and died in combat

1) Player has arrived on death screen
2) Player left-clicks on ”Load Game” button
3) Player chooses from a list of saves

Termination Outcome: Player loads their saved game from the death screen and is returned to the point their saved game was created with all faculties the same as when they saved

Use Case Number: 4

Use Case Name: Movement

Description: Player wishes to move their sprite on the game world or overworld. They will press one of the assigned ”movement keys” (arrows: Up, Down, Left, Right). This will kick off a process to move the character sprite from where they started to where the character wishes to move.

1) Player is in the game world or overworld
2) Player presses one of the movement keys

Termination Outcome: The Player’s sprite is moved from where they started to one space in the direction they wished to move

Use Case Number: 5

Use Case Name: Combat

Description: Combat with Player is initiated. Player wishes to perform a variety of actions like:

1) Attacking with Melee or Ranged
2) Guard / Defend
3) Use Skill
4) Analyze Enemy
5) Wait
6) Escape
7) Autobattle

Player will access these functions by typing the corresponding number into a UI console.

1) Player is within the Combat UI
2) Player enters their preferred action in the lower console
3) Player selects an enemy or companion to attack or use a skill or item on if required. Otherwise, action is performed on themselves (like wait, guard, etc.).

Termination Outcome: Player performs an action in combat, which will continue until Player dies or all of the enemies to the Player die.

Use Case Number: 6

Use Case Name: Talk to NPCs

Description: Player wishes to talk to an NPC, like shopkeepers or story NPCs, or companions. They will press an ”Interact” key while near the NPC. This kicks off the process to push text into a text box for the Player to read and react to

1) Player presses the interact key while one space away from an NPC they wish to talk to
2) NPC text is inserted into a text box for the Player to read

Termination Outcome: Player can read the NPCs text

Alternative: Player is talking to a Shop NPC or a Story NPC

1) Player presses the interact key while one space away from the Shopkeeper NPC or Story NPC

2) A list of choices for the Player appears
3) Player selects a choice (Shop, Talk, etc.)
4) Player is guided to the Shopkeeper's inventory if "Shop" is selected or engages in conversation with the NPC if "Talk" is selected

Termination Outcome: Player has chatted with the NPC or is entered into the Shopkeeper's shop menu

Use Case Number: 7

Use Case Name: Manage Inventory

Description: Player wishes to access their own items or switch their equipped equipment. They will click on an "Inventory" button. This will kick off a process to manage their inventory.

1) Player opens their inventory using a pre-defined "Inventory" hotkey, displaying a menu which holds the Player's currently worn and used equipment, a list of "consumable" items, and a list of unused equipment
2) Player can select on consumable items to use them, can select on equipped items to unequip them, and click on unequipped items to switch to or initially wear

Use Case Number: 8

Use Case Name: Check Character Stats

Description: Player wishes to check their stats and a description of what said stats do. They will open a "Character" menu by pressing a pre-defined hotkey. This will kick of a process to display said menu and allow the character to read a description of their stats by clicking on the names of the stat

1) Player opens their "Character" menu using a certain hotkey
2) "Character" menu displays Player's current stats, health and skill points, and current resistances and weaknesses. Also current status effects.

Termination Outcome: Player can click on stat names to see a description of said stats as well as observe their current status in the form of HP and SP as well as resistances and weaknesses.

Use Case Number: 9

Use Case Name: Manage Companions

Description: Player wishes to manage their companions, by either dismissing, recruiting, or observing companions' stats and skills. They click on a companion portrait. This kicks off a process to access said information and choices

1) Player clicks on a companion portrait to manage them or talks to a dismissed companion
2) A menu is displayed which offers the options to "Dismiss" them from the party or "Recruit them to the party if they were dismissed
3) In this menu, Player can observe said companion's stats and skills, as well as HP, SP, resistances, and weaknesses.

Termination Outcome: Player has managed their companion by observing their status as well as being able to recruit or dismiss them

2.3. Interface Mockups

The following is a mock-up of CYBERNUKE's main menu screen. The main menu is tied to use cases 1 (Start New Game), 3 (Load Game), and use cases for the Options Menu and Exit Game.

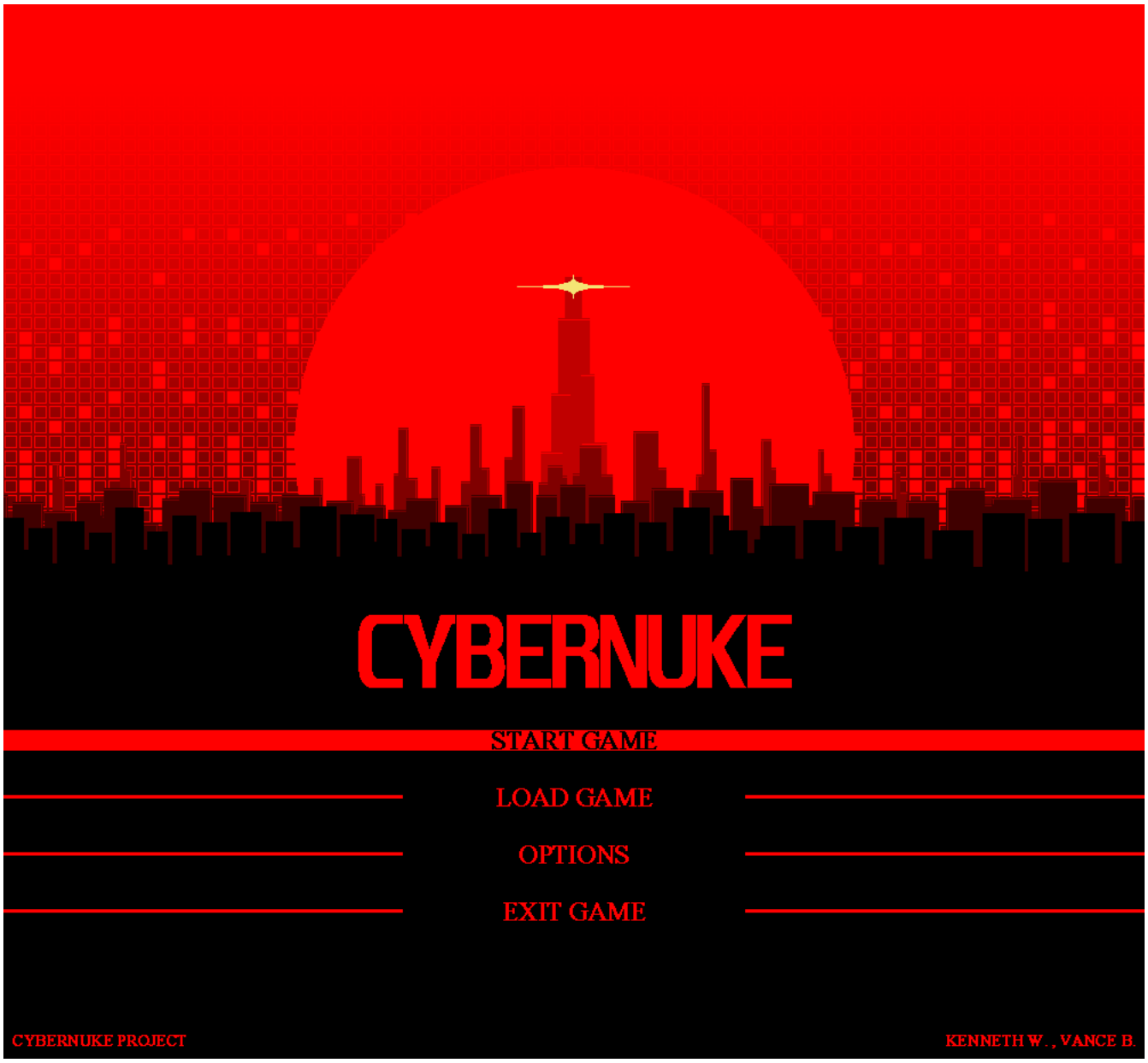

Figure 1. CYBERNUKE Main Menu Screen Mockup

The following is a mock-up of CYBERNUKE's combat screen. The combat screen is tied to use case 5 (Combat).

Figure 2. CYBERNUKE Combat Screen Mockup

3. Project Timeline

1. Requirements

We identified the requirements in the early parts of the project's lifetime (near the beginning of the semester.) Those requirements would later be revised as we realized we were running out of time to implement all of the features we originally planned for.

2. Design

This stage resulted in the creation of the first UML diagram for CYBERNUKE.

3. Implementation

Implementation was started during Spring Break with Kenneth creating the Main Menu and implementing the MVVM pattern for window operation. Kenneth then implemented the Combat Menu and Enemies, and Vance implemented the Character Menu and Characters.

4. Testing

Testing has been done as we worked on implementation due to time constraints with the project.

5. Maintenance

Omitted, project does not need to be maintained after completed.

4. Project Structure

The basis of Cybernuke's structure is built on top of the MVVM (Model-View-ViewModel) pattern. Views in this project are essentially the menus or windows that allow the user to actually play the game. For instance, the MainMenuView is graphically the main menu of the game, and the CombatView is graphically the GUI and functions used in facilitating combat between the player and npcs. The main code and functionality for the views is located in each view's ViewModel.cs code file. Each View will have functionality that allows said view to navigate to other views: for example, the CombatView will be able to navigate to the OverworldView, and vice versa upon finishing and starting combat.

For the MainMenuView, (which is opened as the initial menu available to the player upon launching the game) it must connect to the CutsceneView, the TownView, and the OverworldView through use of the main menu's load menu, as starting the game fresh with a new game or loading a previous saved game will lead into one of these views after data is initialized and loaded.

For the CombatView which will be used during combat with rng-determined battles or scripted boss battles, a list of enemy targets and list of players (main character and companions) is used to track turn order and actions and effects in combat. Character and companion data will be pulled from lists initialized in the MainWindowView (not MainMenuView) and referenced constantly throughout gameplay to track equipment, stats, hp, sp, and so on. Enemy data and the potential enemy lineup during combat will be loaded through file at the beginning of combat based on a randomly generated number system.

For the OverworldView, this will serve as the player's method for moving throughout the game's world and exploring locations and places of interest. This is done through use of a grid-like movement system where the player can move one tile each action to cover distance. During each movement step, chances of a random encounter will be factored depending on the location the player is in (for instance, the player will not have random encounters in a safe-designated area while having lots of encounters in "dungeons").

From the OverworldView, the player can open the PauseMenuView, which functions to allow the player to examine skills, character info, and the more general map (of the entire world, not where the player currently is and is moving on), as well as configuring options, closing the View, and exiting the game.

For the TownView (accessed upon reaching a town), the user will use a sequence of buttons to access features the town holds. For example, to visit certain NPCs, clickable buttons will be used to do that. To visit shops and leave the town, buttons will be used for that as well. The ability to save and load game is available here as well.

For the CutsceneView, this is essentially using premade videos to deliver exposition and important story events to the player through use of a content presenter on the MainWindow that is hidden after the cutscene finishes and control is granted back to the player.

4.1. UML Outline

4.2. Design Patterns Used

1. Prototype Design Pattern

We used a Prototype for the enemies in CYBERNUKE. When the Combat Menu is opened it will be given a data file that contains which enemies to spawn. It will then take the data files for those enemies and instantiate an Enemy UserControl Object to display on the Combat Menu.

2. Decorator Design Pattern

We used a Decorator for the Pause Menu in CYBERNUKE. When opening the Pause Menu it unhides a Content Presenter that is present on the MainWindow. This Content Presenter holds the Pause Menu and renders over the underlying window so that the user cannot accidentally activate anything while paused.

3. MVVM (Model-View-ViewModel) Design Pattern

We used the MVVM design pattern for CYBERNUKE. We chose to use it as it would allow the program to seamlessly switch between menus without breaking the user's immersion in the game. It also allows us to store data in MainWindow as it stays open persistently.

5. Results

5.1. Future Work

For early deliverables, what are your next steps? (HINT: you will typically want to look back at your timeline and evaluate: did you meet your expected goals? Are you ahead of schedule? Did you decide to shift gears and implement a new feature?) By the end, what do you plan on doing with this project? Will you try to sell it? Set it on fire? Link to it on your resume and forget it exists?

References

[1] Britannica, T. Editors of Encyclopaedia. *"cyberpunk."* Encyclopedia Britannica, 1999. https://www.britannica.com/art/cyberpunk.

[2] Wiktionary. *"roguelite"* https://en.wiktionary.org/wiki/rogue-lite

[3] whatNerd, Joel Lee. *"What's a Roguelike vs. Roguelite? Is the Difference Really That Important?"* whatNerd, 2021. https://whatnerd.com/what-is-a-roguelike-roguelite-difference/.